

CS604-PRACTCAL LAB SOLUTION: BY VU-GATEWAY!

LESSON 1 : WEEK 1 LAB-01

Installation of Virtual Box and Ubuntu

LESSON 02 : WEEK 02 LAB-02

Linux commands for managing directories and files

Practice the following Linux commands in Shell:

1. Write a command to show your current working directory.
2. Write a command to change your current directory to Desktop.
3. Write a command to create a directory structure as: CS604/Files/CFiles
4. Write a command to display the contents of Desktop.
5. Write a command to remove a directory named CS604.

SOLUTION:

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

- ✚ **Show your current working directory: `pwd`**
- ✚ **Change your current directory to Desktop: `cd ~/Desktop`**
- ✚ **Create a directory structure as CS604/Files/CFiles:**
`mkdir -p CS604/Files/CFiles`
- ✚ **Display the contents of Desktop: `ls ~/Desktop`**
- ✚ **Remove a directory named CS604: `rm -r CS604`**

LESSON 03 : WEEK 03

LAB-03

• Inter-process communication between processes using PIPES

Write a program in C for inter-process communication between child and parent processes. You have to use pipe() system calls for creating a pipe between parent and child processes. The child will write the student id through PIPE and parent will read the student id from pipe and display it on screen. Parent will wait for child process until child write data to pipe. Compile and execute your program in Command prompt to display the output as shown below.

```
instructor@cs604:~/Desktop$ ./pipeProgram
Enter your ID: BC123456789
Stuent ID: BC123456789instructor@cs604:~/Desktop$
```

Note: Use the read() and write() system call for reading from the pipe and writing to the Pipe.

See the following link for read() and write() system calls

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

SOLUTION:

```
#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

    int pipefd[2]; // Array to hold pipe descriptors
    pid_t pid; // Process ID
    char write_msg[100]; // Message to write into the pipe
    char read_msg[100]; // Buffer to read message from the pipe

    // Create a pipe
    if (pipe(pipefd) == -1) {
        perror("Pipe creation failed");
        return 1;
    }

    pid = fork(); // Create a child process
    if (pid < 0) {
        perror("Fork failed");
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
return 1;
}

if (pid == 0) { // Child process
    close(pipefd[0]); // Close the read end of the pipe
    printf("Enter your ID: ");
    fgets(write_msg, sizeof(write_msg), stdin); // Read input from user
    write(pipefd[1], write_msg, strlen(write_msg) + 1); // Write to pipe
    close(pipefd[1]); // Close the write end after writing
} else { // Parent process
    close(pipefd[1]); // Close the write end of the pipe
    read(pipefd[0], read_msg, sizeof(read_msg)); // Read from pipe
    printf("Student ID: %s", read_msg); // Display the message
    close(pipefd[0]); // Close the read end after reading
}

return 0;
}
```

□ Explanation of the Code:

1. **pipe(pipefd)**: Creates a pipe. pipefd[0] is for reading, and pipefd[1] is for writing.
2. **fork()**: Creates a child process.

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

- In the **child process**:
 - Closes the read end of the pipe (pipefd[0]).
 - Accepts input from the user (fgets).
 - Writes the input into the pipe using write().
 - Closes the write end after writing.
- In the **parent process**:
 - Closes the write end of the pipe (pipefd[1]).
 - Reads the input from the pipe using read().
 - Displays the input to the screen.
 - Closes the read end after reading.

Steps to Compile and Execute:

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

1. Save the code in a file, e.g., `pipeProgram.c`.
2. Compile the program using:

```
bash  
  
gcc -o pipeProgram pipeProgram.c
```

3. Run the program:

```
bash  
  
./pipeProgram
```

Sample Output:

```
bash  
  
Enter your ID: BC123456789  
Student ID: BC123456789
```

LESSON 04 : WEEK 04

LAB-04

Threads creation and termination

Write a program in C that will create two threads named as Thread 1 and Thread 2. You are required to run these threads in parallel fashion and display the Thread IDs

Join Whatsapp Group Link: 📌 📌

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

according to the output as shown below. At the end, all threads will be terminated. Compile & run C program on Linux Operating system.

```
instructor@cs604: ~/Desktop
instructor@cs604:~$ cd Desktop
instructor@cs604:~/Desktop$ gcc -pthread lab4.c -o lab4
instructor@cs604:~/Desktop$ ./lab4
Thread with ID: 3075423040 created
Thread with ID: 3067030336 created

Threads are going to be terminated one by one
instructor@cs604:~/Desktop$
```

FIFO Inter-Process Communication

Inter-process communication using FIFO, Foreground, background processs, read(), write() system call.

FIFO example

Write the two C programs named as program1 and program2 that perform inter-process communication through FIFO. Functionality of the two programs should be as follows.

program1 should create a FIFO as fifo_one by using mknod() system call or through mkfifo() library call. program1 should open the fifo_one for reading purpose through open() system call. Then use the read() system call to read from fifo_one, your name and VUID that is written by program2 in fifo_one. After reading your name and VUID, program1 should display it on the screen by write() system call.

program2 should open the fifo_one for writing purpose through open() system call. Then use the write() system call to write in fifo_one your name and VUID. And finally close and remove the fifo_one.

Use only read() and write() system call for reading and writing your name and VUID from fifo and finally writing on terminal screen . As by default in FIFO both read() and write() system call do blocking I/O.

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION: BY VU-GATEWAY!

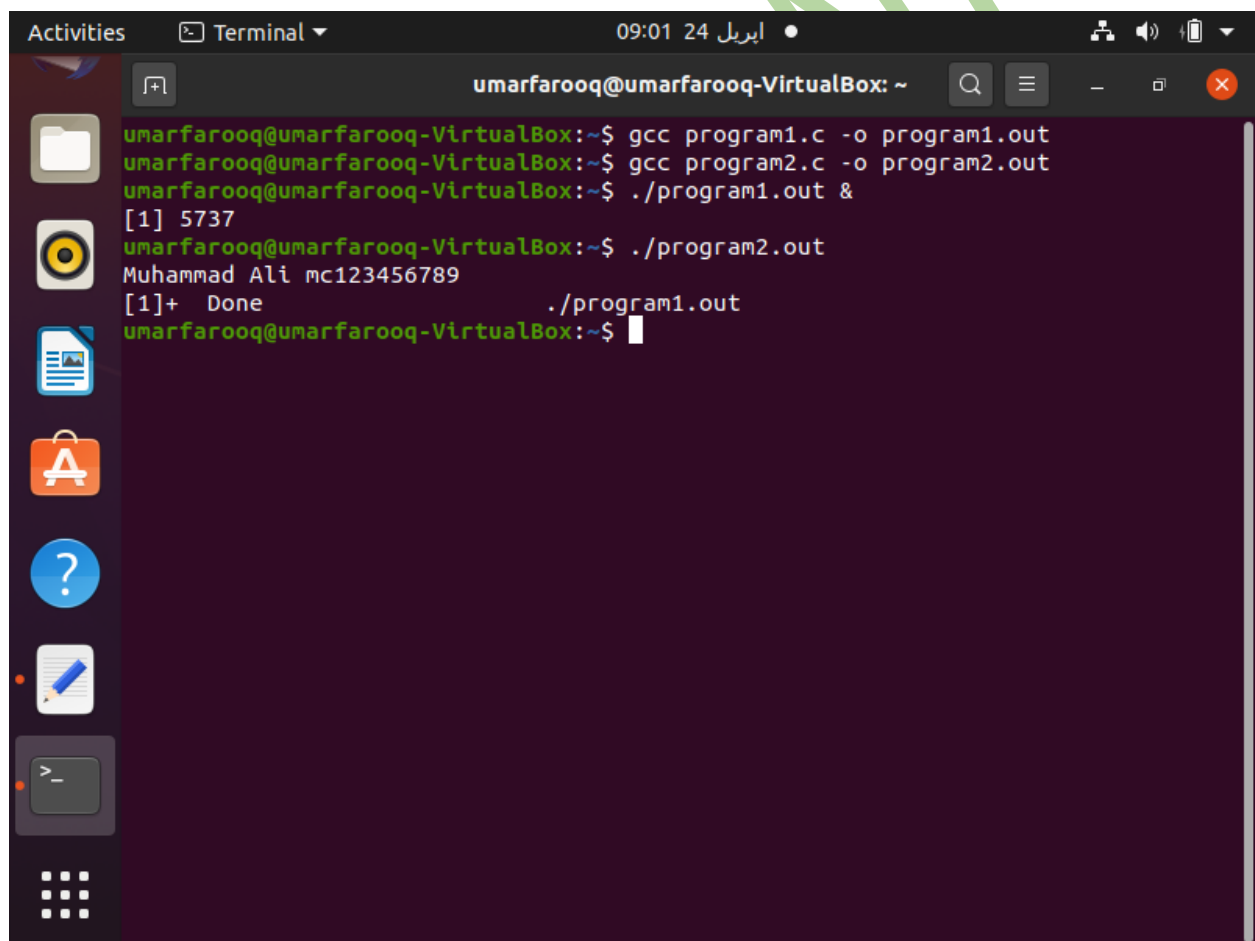
As the two program can't be run simultaneously in foreground on terminal window. So you need to run the program1 in background and subsequently program2 should be run in foreground.

Following is the sample screenshot of both program. You need to perform the following steps in screen shot.

In screenshot firstly you should compile the both programs.

Run the program1 in background.

Finally run the program2.



```
umarfaroq@umarfaroq-VirtualBox: ~  
umarfaroq@umarfaroq-VirtualBox:~$ gcc program1.c -o program1.out  
umarfaroq@umarfaroq-VirtualBox:~$ gcc program2.c -o program2.out  
umarfaroq@umarfaroq-VirtualBox:~$ ./program1.out &  
[1] 5737  
umarfaroq@umarfaroq-VirtualBox:~$ ./program2.out  
Muhammad Ali mc123456789  
[1]+  Done                  ./program1.out  
umarfaroq@umarfaroq-VirtualBox:~$
```

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION: BY VU-GATEWAY!

SOLUTION:

Code program1.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
extern int  errno;
#define FIRSTFIFO  "/tmp/fifo_one"

#define PERMS  0666
#define NAME_VUID  "Muhammad Ali mc123456789\n"
int main() {
    char buff[1024];
    int fdread, fdwrite;
    int n, size;
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
if ((mknod (FIRSTFIFO, S_IFIFO | PERMS, 0) < 0) && (errno != EEXIST))
{
    perror ("mknod FIFO1");
    exit (1);
}
if ((fdread = open(FIRSTFIFO, 0)) < 0)
{
    perror ("open First FIFO");
    exit (1);
}
size = strlen(NAME_VUID) + 1;
if ((n = read(fdread, buff, size)) < 0)
{
    perror ("First Progarm read");
    exit (1);
}
if (write (1, buff, n) < n)
{
    perror("First Program write1"); exit (1);
}
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
close (fdread);  
return 0;  
}
```

Code for program2.c

```
#include <stdio.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/errno.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <unistd.h>  
extern int  errno;  
#define FIRSTFIFO  "/tmp/fifo_one"  
#define PERMS  0666  
#define NAME_VUID  "Muhammad Ali mc123456789\n"  
int main()  
{  
    char buff[BUFSIZ];  
    int fdread, fdwrite, n, size;  
    if ((fdwrite = open(FIRSTFIFO, 1)) < 0)
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
{  
    perror ("Second program open first FIFO");  
    exit (1);  
}  
  
size = strlen(NAME_VUID) + 1;  
if (write(fdwrite, NAME_VUID, size) != size)  
{  
    perror ("Second Program write1"); exit (1);  
}  
  
close(fdwrite);  
/* Remove FIFO as we are done with it */  
if (unlink (FIRSTFIFO) < 0)  
{  
    perror("client unlink FIFO1");  
    exit (1);  
}  
    exit (0);  
return 0;  
}
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION: BY VU-GATEWAY!

LESSON 05: WEEK 05 LAB-05

First Come First Serve is Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time.

Implementing FCFS scheduling algorithm in C

Write a program in C to implement FCFS scheduling algorithm. Given the list of Processes and their burst time, calculate waiting time and turnaround time for each process. Also calculate average waiting time and average turnaround time. For example, the list of processes and their CPU burst time are as follows:

Processes	Burst Time
P0	10
P1	4
P2	6
P3	8

Assume that all the processes arrive at time 0 in sequence P0, P1, P2 P3

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
instructor@cs604:~/Desktop$ gcc FCFS.c -o FCFS
instructor@cs604:~/Desktop$ ./FCFS
P0: Wait Time: 0      Turnaround Time: 10
P1: Wait Time: 10     Turnaround Time: 14
P2: Wait Time: 14     Turnaround Time: 20
P3: Wait Time: 20     Turnaround Time: 28

Average Wait time: 11

Average turnaround time: 18

instructor@cs604:~/Desktop$
```

SOLUTION:

Solution:

```
#include <stdio.h>

int main(){
    int n = 4 ;      // number of processes
    int bTime[4] ;   // array for storing burst time for each process
    int wTime[4] ;   // array for storing wait time for each process
    int tATime[4] ;  // array for storing turnaround time for each process
    bTime[0] = 10 ;
    bTime[1] = 4 ;
    bTime[2] = 6 ;
    bTime[3] = 8 ;
    int avgWTime = 0 ;
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
int avgtATime = 0 ;
wTime[0] = 0 ;
// loop for calculating waiting time for each process
for(int i = 1; i<n; i++){
    wTime[i] = wTime[i-1]+bTime[i-1] ;    // calculating wait time for each process
    avgWTime = avgWTime + wTime[i] ; }
// loop for calculating turnaround time for each process
for(int i = 0; i<n; i++){
    tATime[i] = wTime[i]+bTime[i] ;    // calculating turnaround time for each
process
    avgtATime = avgtATime + tATime[i] ;
}
avgWTime = avgWTime / n; // calculating average waiting time
avgtATime = avgtATime / n ; // calculating average turnaround time
// loop for displaying waiting time and turnaround times
for (int i=0; i<n; i++){
    printf("P%i: ", i) ;
    printf("Wait Time: %i\t", wTime[i]) ;    // printing wait time for each process
    printf("Turnaroung Time: %i\n", tATime[i]) ; // printing turnaround time for each
process
}
// displaying average waiting time and average turnaround time
```

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
printf("\nAverage Wait time: %i\n", avgWTime) ;  
printf("\nAverage Wait time: %i\n\n", avgtATime) ;  
} // end of main()
```

See the following link for different process Scheduling Algorithms.

LESSON 06: WEEK 06

LAB-06

Implementing Round Robin scheduling algorithm in C

Write a program in C to implement the Round Robin scheduling algorithm. The time slice for each process is 5 units. Given the list of Processes and their CPU burst time, calculate the waiting time and turnaround time for each process. You also have to calculate average waiting and average turnaround time and display their values on the screen.

For example, the list of processes and their CPU burst time are as follows:

Processes	Burst Time
P0	12
P1	9
P2	4
P3	6

Time slice = 5 units

Assume that all the processes arrive at time 0 in sequence P0, P1, P2 P3

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

Output of program:

```
instructor@cs604:~/Desktop$ gcc RR.c -o RR
instructor@cs604:~/Desktop$ ./RR

Turnaround Time: 31
Turnaround Time: 28
Turnaround Time: 14
Turnaround Time: 29
Waiting Time: 19
Waiting Time: 19
Waiting Time: 10
Waiting Time: 23
Average Turnaround Time: 25
Average Waiting Time: 17
instructor@cs604:~/Desktop$
```

SOLUTION:

```
#include <stdio.h>

int main(){
    int n = 4 ;
    int avgWTime = 0, avgTTime = 0 ;
    int timeSlice= 5 ;
    int tSlice[4] ;
    int bTime[4] ;
    int tATime[4];
    int wTime[4];
    int count[4] ;
    bTime[0] = 12 ;
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
bTime[1] = 9 ;
bTime[2] = 4 ;
bTime[3] = 6 ;
int value[4] ;
for (int i = 0 ; i < n ; i++ ){
    count[i] = 0 ;
    value[i] = 0 ;
}
int flag ;
int counter = 0 ;
do{
    flag = 0 ;
    for ( int i = 0 ; i < n ; i++ ){
        if (bTime[i] < timeSlice || bTime[i] == 0)
            tSlice[i] = bTime[i];
        else
            tSlice[i] = timeSlice ;
        for (int j = 0 ; j < tSlice[i] ; j++ ){
            bTime[i]-- ;
            counter++ ;
        }
    }
}
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION: BY VU-GATEWAY!

```
tSlice[i] = bTime[i];
if(value[i] != 1)
{
    count[i] = counter ;
}
}
for ( int i = 0 ; i<n; i++ ){
    if (bTime[i] != 0){
        flag = 1 ;
    }
    else
        value[i] = 1;
}
}while (flag != 0);
bTime[0] = 12 ;
bTime[1] = 9 ;
bTime[2] = 4 ;
bTime[3] = 6 ;
for (int i = 0 ; i<n ; i++ ){
    wTime[i] = count[i] - bTime[i] ;
}
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTCAL LAB SOLUTION:

BY VU-GATEWAY!

```
printf("\n");
// for loop to display turnaround time for each process
for (int i = 0 ; i<n ; i++ ){
    printf("Turnaround Time: %i\n", count[i]) ;
}
// for loop to display wait time for each process
for (int i = 0 ; i<n ; i++ ){
    printf("Waiting Time: %i\n", wTime[i]) ;
}
// for loop to calculate average wait time
for (int i = 0 ; i<n ; i++ ){
    avgWTime = avgWTime+wTime[i] ;
}
// for loop to calculate average turnaround time
for (int i = 0 ; i<n ; i++ ){
    avgTTime = avgTTime+count[i] ;
}
printf("Average Turnaround Time: %i\n", avgTTime/n) ;
printf("Average Waiting Time: %i\n", avgWTime/n) ;
} // end of main()
```

Join Whatsapp Group Link:  

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>

CS604-PRACTICAL LAB SOLUTION: BY VU-GATEWAY!

Round Robin Algorithm falls under the category of preemptive scheduling.

For More information about Process Scheduling Algorithm (Round Robin), See the following link.

VU-GATEWAY

Join Whatsapp Group Link: 🖱️ 🖱️

<https://chat.whatsapp.com/GecIrau2Nit0D4F5DjDiGM>