



# **D.A.V. College**

**Affiliated to Tribhuvan University**

**Dhobighat, Lalitpur**

## **LAB REPORT FOR OPERATING SYSTEM**

### **SUBMITTED BY:**

Salin Maharjan

Roll No: 16

### **SUBMITTED TO:**

**Manil Vaidhya**

LECTURER

# INDEX PAGE

[illegible]

# LAB 0: INTRODUCTION TO BATCH AND SHELL COMMANDS

## OBJECTIVE:

The primary objective of this lab report is to introduce students to the fundamental concepts and practical applications of batch and shell commands in the context of operating systems. And aims to familiarize students with batch and shell commands, enhancing their ability to navigate and manipulate the command line

## THEORY:

### Introduction to Batch Commands:

Batch Scripting consists of a series of commands to be executed by the command-line interpreter, stored in a plain text file. It is not commonly used as a programming language and so it is not commonly practiced and is not trending but its control and dominance in the Windows environment can never be neglected. Almost every task and every action can be performed and executed by a simple sequence of commands typed on the Windows Command Prompt.

### Introduction to Shell Commands:

The shell is the command interpreter on the Linux systems. It is a program that interacts with the users in the terminal emulation window. Shell commands are instructions that instruct the system to do some action. Linux shell commands are a set of instructions that can be executed in a command-line interface (CLI) or terminal. They are used to interact with the shell environment and perform various tasks such as navigating the file system, managing files and directories, and executing programs.

### Some Basic Linux Shell Commands:

1. pwd- It give the information of current dir
2. ls - It lists out the file and directories in the current directory
3. cd-It is used to Change directory
4. mkdir - It is used to make a directory

5. rm - It is used to remove files or directory. If the folder contains some files then use rm -r to delete the directory
6. touch - It is used to create a new file.
7. man - Shows the manual pages of the command.
8. cp-It is used to copy the files through the command line. First you have to give the location of the current directory and destination directory.
9. mv - It is used to move the files through the command line. First you have to give the location of the current directory and destination directory.
10. cat - It is used to display the contents of the files.

Question 1: Write basic batch commands and display output

Solution:

basiccommands.bat file

@ECHO OFF

ECHO =====

ECHO Section 1: Hardware Info

systeminfo | findstr /C:"OS Name"

systeminfo | findstr /C:"OS Version"

ECHO =====

ECHO =====

ECHO Section 2: Network Info

ipconfig

tracert www.google.com

ping www.google.com

ECHO =====

PAUSE

Output:

```
=====
Section 1: Hardware Info
OS Name:                Microsoft Windows 11 Home Single Language
OS Version:             10.0.22621 N/A Build 22621
BIOS Version:           American Megatrends International, LLC. E15K1IMS.30D, 26-06-2023
=====
```

```
=====
Section 2: Network Info

Windows IP Configuration

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . :
    Link-local IPv6 Address . . . . . : fe80::4023:f4c8:c384:94ec%13
    IPv4 Address. . . . . : 192.168.1.71
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.254

Tracing route to www.google.com [142.250.193.132]
over a maximum of 30 hops:

  1    3 ms    8 ms    1 ms    dsldevice.lan [192.168.1.254]
  2    6 ms    2 ms    2 ms    10.16.0.2
  3   50 ms   45 ms   46 ms    nsg-corporate-185.75.186.122.airtel.in [122.186.75.185]
  4    *     41 ms    *     116.119.109.97

  5   46 ms   39 ms   41 ms   142.250.169.206
  6   44 ms   43 ms   44 ms   142.250.208.105
  7   41 ms   41 ms   38 ms   142.251.55.227
  8   46 ms   44 ms   43 ms   maa05s25-in-f4.1e100.net [142.250.193.132]

Trace complete.

Pinging www.google.com [142.250.193.132] with 32 bytes of data:
Reply from 142.250.193.132: bytes=32 time=47ms TTL=116
Reply from 142.250.193.132: bytes=32 time=48ms TTL=116
Reply from 142.250.193.132: bytes=32 time=44ms TTL=116
Reply from 142.250.193.132: bytes=32 time=46ms TTL=116

Ping statistics for 142.250.193.132:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 44ms, Maximum = 48ms, Average = 46ms
=====
Press any key to continue . . .
```

Question 1: Write basic batch commands and display output

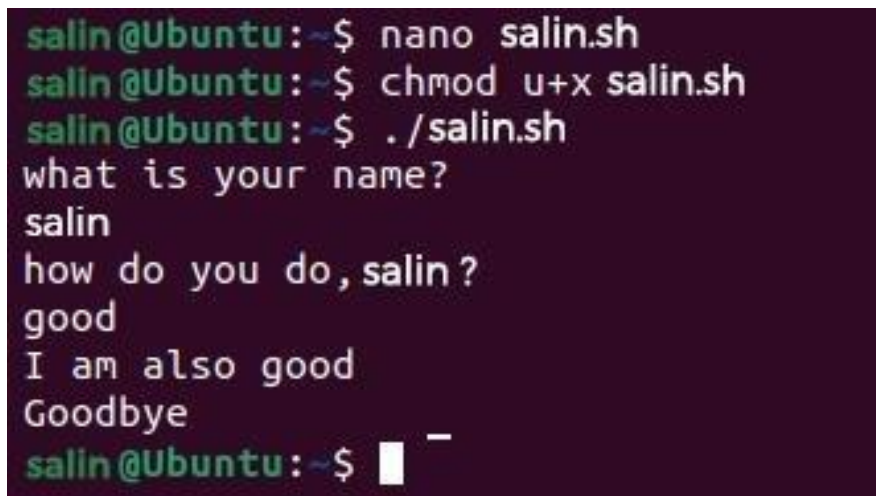
Solution:

Salin.sh file

A screenshot of a nano text editor window titled 'salin.sh'. The window has a menu bar with 'Open', 'Save', and a hamburger menu icon. The file content is as follows:

```
1#!/bin/sh
2echo "what is your name?"
3read name
4echo "how do you do, $name?"
5read remark
6echo "I am also $remark"
7echo "Goodbye"
```

Output:

A screenshot of a terminal window showing the execution of the salin.sh script. The prompt is 'salin@Ubuntu:~\$'. The user enters 'nano salin.sh', then 'chmod u+x salin.sh', and finally './salin.sh'. The script's output is displayed:

```
salin@Ubuntu:~$ nano salin.sh
salin@Ubuntu:~$ chmod u+x salin.sh
salin@Ubuntu:~$ ./salin.sh
what is your name?
salin
how do you do, salin?
good
I am also good
Goodbye
salin@Ubuntu:~$
```

CONCLUSION:

In conclusion, this lab provided a solid foundation for understanding and utilizing batch and shell commands, empowering us to navigate the command-line interface with confidence and efficiency.

## LAB 1: INTRODUCTION TO PROCESS COMMANDS

### OBJECTIVE:

The objective of this lab report is to gain a comprehensive understanding of process identification (PID), parent process identification (PPID), and the fork system call in Unix-like operating systems.

### THEORY:

- **PID (Process ID):**

A PID, or Process ID, is a unique numerical identifier assigned to each running process in a Unix-like operating system. PIDs are crucial for tracking and managing processes. They are used to differentiate between different processes and are assigned sequentially as new processes are created.

- **PPID (Parent Process ID):**

PPID, or Parent Process ID, refers to the PID of the parent process that spawned a particular process. When a new process is created, it inherits its PPID from the process that created it. The PPID is essential for establishing the hierarchical relationship between processes.

- **Fork System Call:**

The fork system call is a fundamental mechanism for process creation in Unix-like operating systems. When a process calls fork, a new process is created, which is referred to as the child process. The child process is an almost identical copy of the parent process.

Program 1: Write a program demonstrating use of PID

### Solution:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

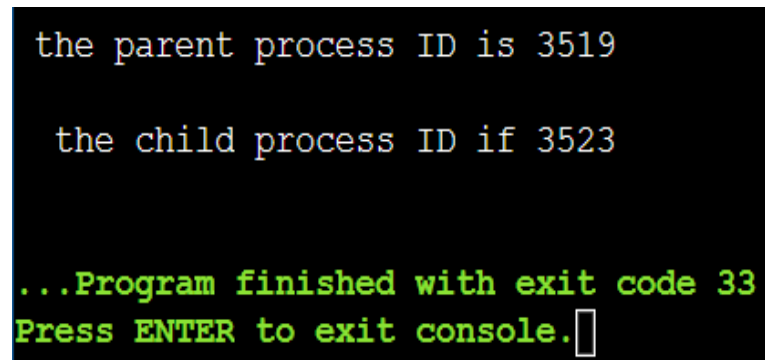
void main(){
```

```

int pid, pid1, pid2;
pid=fork();
if (pid==-1){
    printf("ERROR IN PROCESS CREATION \n");
    exit(1);
}
if(pid!=0){
    pid1=getpid();
    printf("\n the parent process ID is %d \n", pid1);
}else{
    pid2= getpid();
    printf("\n the child process ID if %d \n", pid2);
}
}

```

Output:



```

the parent process ID is 3519

the child process ID if 3523

...Program finished with exit code 33
Press ENTER to exit console.

```

Program 2: Write a program demonstrating use of PPID

Solution:

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
    int pid, pid1, pid2;

```



```

pid=fork();
if (pid==0){
    sleep(3);
    printf("child[1]--> pid = %d and ppid = %d\n", getpid(), getppid());
}else{
    pid1= fork();
    printf("\n the child process ID is %d \n", pid2);
    if (pid1==0){
        sleep(2);
        printf("child[2]--> pid = %d and ppid = %d\n", getpid(), getppid());
    }else{
        pid2 = fork();
        if (pid2 == 0){
            printf("child[3]--> pid = %d and ppid = %d\n", getpid(), getppid());
        }else{
            sleep(3);
            printf("parent --> pid = %d\n", getpid());
        }
    }
}
return 0;
}

```

Output:

```

child[3]--> pid = 2629 and ppid = 2623
child[2]--> pid = 2628 and ppid = 2623
parent --> pid = 2623
child[1]--> pid = 2627 and ppid = 2623

```

```

...Program finished with exit code 0
Press ENTER to exit console.

```

## CONCLUSION:

In conclusion, the concepts of PID (Process ID), PPID (Parent Process ID), and the fork system call play pivotal roles in the functioning and management of processes within Unix-like operating systems. PID serves as a unique identifier for each running process, facilitating process tracking and resource allocation.

## LAB 2: Process Scheduling Algorithms

### OBJECTIVE:

The report aims to provide a clear understanding of how each process scheduling algorithm functions, examining their advantages and limitations. Through a comparative analysis, we will investigate the impact of these algorithms on system performance metrics such as turnaround time, waiting time, and throughput.

### THEORY:

Process scheduling algorithms are integral components of operating systems that manage the execution of multiple processes, ensuring efficient resource utilization and system responsiveness. And they are:

- First-Come-First-Serve (FCFS) Algorithm:

In the FCFS algorithm, processes are executed in the order they arrive, with the first process that enters the ready queue being the first one to be executed. This simplistic approach is easy to understand and implement. However, FCFS may lead to the "convoy effect," where shorter processes get delayed behind longer ones, impacting overall system efficiency. The primary advantage of FCFS lies in its simplicity and ease of implementation, making it suitable for scenarios with low computational overhead.

- Shortest Job First (SJF) Algorithm:

Contrastingly, the SJF algorithm prioritizes the execution of the shortest job first, aiming to minimize the total processing time. This algorithm requires predicting the burst time of each process, which can be challenging in real-world scenarios. Preemptive SJF allows for dynamic adaptation to changing burst times, while non-preemptive SJF commits to a selected process until completion. SJF is known for its efficiency in minimizing waiting times and maximizing throughput, especially in scenarios where burst times are accurately known.

- Priority Algorithm:

A priority algorithm is a computational approach used in various fields to determine the order or sequence in which tasks, processes, or elements should be executed or processed based on assigned priorities. The fundamental concept behind priority algorithms is to allocate resources or attention to higher-priority items before lower-priority ones.

Program 1: Write a program demonstrating use of FCFS Algorithm for programs having same arrival time

Solution:

```
#include <stdio.h>
int main(){
    int n, bt[20], wt[20], tat[20], avwt=0, avtat=0, i, j;
    printf("Enter total number of processes (maximim 20): ");
    scanf("%d", &n);
    printf("\n Enter the process burst time \n");
    for (i = 0; i < n; i++) {
        printf("P[%d]:", i+1);
        scanf("%d", &bt[i]);
    }
    wt[0]=0;
    for (i = 1; i < n; i++) {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }
    printf("\n Process \t\t Burst Time \t Waiting Time\t Turnaround Time");
    for(i=0;i<n;i++){
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+= tat[i];
        printf("\n P[%d] \t\t %d \t\t %d \t\t %d", i+1, bt[i], wt[i], tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\n Average Turnaround Time: %d", avtat);
    printf("\n Average Waiting Time: %d", avwt);
    return 0;
}
```

Output:

```
Enter total number of processes (maximim 20): 5

Enter the process burst time
P[1]:4
P[2]:2
P[3]:3
P[4]:2
P[5]:1

Process          Burst Time      Waiting Time    Turnaround Time
P[1]              4                0                4
P[2]              2                4                6
P[3]              3                6                9
P[4]              2                9               11
P[5]              1               11               12
Average Turnaround Time: 8
Average Waiting Time: 6
```

Program 2: Write a program demonstrating use of FCFS Algorithm for programs having different arrival time

Solution:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main(){
    char pn[10][10],t[10];
    int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp;
    int totwt = 0, tottat = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for(i=0;i<n;i++){
        printf("Enter the Process Name Arrival Time $ Burst Time: ");
        scanf("%s%d%d", &pn[i], &arr[i], &bur[i]);
    }
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(arr[i]<arr[j]){
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                temp = bur[j];
                bur[i] = bur[j];
                bur[j] = temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    }
    for(i=0;i<n;i++){
        if(i==0){
            star[i] = arr[i];
        }else{
            star[i] = finish[i-1];
        }
        star[i] = arr[i];
        wt[i] = star[i] - arr[i];
        finish[i] = star[i] + bur[i];
        tat[i] = finish[i] - arr[i];
    }

    printf("\n Pname Arrtime Burttime Waittime Start TAT Finish");
    for(i=0;i<n;i++){
        printf("\n%s \t%3d \t%3d \t%3d \t%3d \t%6d \t%6d", pn[i], bur[i], wt[i], star[i], tat[i],
finish[i]);
        totwt += wt[i];
        tottat += tat[i];
    }
}
```

```

    }
    printf("\n Average Waiting Time: %f", (float)totwt);
    printf("\n Average Turn Around Time: %f", (float)tottat);
    return 0;
}

```

Output:

```

Enter total number of processes (maximum 20): 5

Enter the process burst time and arrival time:
P[1] Burst Time:4
P[1] Arrival Time:2
P[2] Burst Time:3
P[2] Arrival Time:2
P[3] Burst Time:1
P[3] Arrival Time:2
P[4] Burst Time:3
P[4] Arrival Time:4
P[5] Burst Time:5
P[5] Arrival Time:6

Process Burst Time      Arrival Time      Waiting Time      Turnaround Time
P[1]      4              2                  0                  4
P[2]      3              2                  2                  5
P[3]      1              2                  5                  6
P[4]      3              4                  4                  7
P[5]      5              6                  5                  10

Average Turnaround Time: 6
Average Waiting Time: 3

```

Program 3: Write a program demonstrating use of SJF Algorithm for programs having same arrival time

Solution:

```

#include<stdio.h>
int main() {
    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("\nEnter Burst Time:\n");
    for (i = 0; i < n; i++) {

```

```

    printf("P[%d]:", i + 1);
    scanf("%d", &bt[i]);
    p[i] = i + 1; }
for (i = 0; i < n; i++) {
    pos = i;
    for (j = i + 1; j < n; j++) {
        if (bt[j] < bt[pos])
            pos = j;}
    temp = bt[i];
    bt[i] = bt[pos];
    bt[pos] = temp;
    temp = p[i];
    p[i] = p[pos];
    p[pos] = temp; }
wt[0] = 0;
for (i = 1; i < n; i++) {
    wt[i] = 0;
    for (j = 0; j < i; j++)
        wt[i] += bt[j];
    total += wt[i]; }
avg_wt = (float) total / n;
total = 0;
printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
for (i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    total += tat[i];
    printf("\nP%d\t\t%d\t\t%d\t\t\t\t\t", p[i], bt[i], wt[i], tat[i]);}
avg_tat = (float) total / n;
printf("\n\nAverage Waiting Time = %f", avg_wt);
printf("\n\nAverage Turnaround Time = %f\n", avg_tat);
return 0;}

```

Output:

```

Enter number of process:5

Enter Burst Time:
P[1]:4
P[2]:2
P[3]:3
P[4]:2
P[5]:1

Process  Burst Time      Waiting Time      Turnaround Time
p5          1              0                1
p2          2              1                3
p4          2              3                5
p3          3              5                8
p1          4              8               12

Average Waiting Time=3.400000
Average Turnaround Time=5.800000

```

Program 4: Write a program demonstrating use of SJF Algorithm for programs having different arrival time

Solution:

```
#include<stdio.h>
int main() {
    int bt[20], at[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("\nEnter Burst Time and Arrival Time:\n");
    for (i = 0; i < n; i++) {
        printf("P[%d] Burst Time:", i + 1);
        scanf("%d", &bt[i]);
        printf("P[%d] Arrival Time:", i + 1);
        scanf("%d", &at[i]);
        p[i] = i + 1;}
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
            if (at[j] < at[pos] || (at[j] == at[pos] && bt[j] < bt[pos]))
                pos = j; }
        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;
        temp = at[i];
        at[i] = at[pos];
        at[pos] = temp;
        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;}
    wt[0] = 0;
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        total += wt[i];}
    avg_wt = (float) total / n;
    total = 0;
    printf("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d", p[i], bt[i], at[i], wt[i], tat[i]);}
    avg_tat = (float) total / n;
    printf("\n\nAverage Waiting Time = %f", avg_wt);
    printf("\n\nAverage Turnaround Time = %f\n", avg_tat);
    return 0;}
```



Output:

```
Enter number of processes: 5

Enter Burst Time and Arrival Time:
P[1] Burst Time:4
P[1] Arrival Time:1
P[2] Burst Time:6
P[2] Arrival Time:8
P[3] Burst Time:4
P[3] Arrival Time:6
P[4] Burst Time:2
P[4] Arrival Time:3
P[5] Burst Time:5
P[5] Arrival Time:2

Process Burst Time      Arrival Time      Waiting Time      Turnaround Time
P1          4           1             0                4
P5          5           2             4                9
P4          2           3             9               11
P3          4           6            11               15
P2          6           8            15               21

Average Waiting Time = 7.800000
Average Turnaround Time = 12.000000
```

Program 5: Write a program demonstrating use of SJF Algorithm for programs having different arrival time

Solution:

```
#include<stdio.h>
int main() {
    int bt[20], at[20], p[20], wt[20], tat[20], priority[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("\nEnter Burst Time, Arrival Time, and Priority:\n");
    for (i = 0; i < n; i++) {
        printf("P[%d] Burst Time:", i + 1);
        scanf("%d", &bt[i]);
        printf("P[%d] Arrival Time:", i + 1);
        scanf("%d", &at[i]);
        printf("P[%d] Priority:", i + 1);
        scanf("%d", &priority[i]);
        p[i] = i + 1;
    }
```

```

}
// Sorting based on Arrival Time and Priority
for (i = 0; i < n; i++) {
    pos = i;
    for (j = i + 1; j < n; j++) {
        if (at[j] < at[pos] || (at[j] == at[pos] && priority[j] < priority[pos]))
            pos = j;
    }
    temp = bt[i];
    bt[i] = bt[pos];
    bt[pos] = temp;
    temp = at[i];
    at[i] = at[pos];
    at[pos] = temp;
    temp = priority[i];
    priority[i] = priority[pos];
    priority[pos] = temp;
    temp = p[i];
    p[i] = p[pos];
    p[pos] = temp;
}
wt[0] = 0;
for (i = 1; i < n; i++) {
    wt[i] = 0;
    for (j = 0; j < i; j++)
        wt[i] += bt[j];
    total += wt[i];
}
avg_wt = (float) total / n;
total = 0;
printf("\nProcess\tBurst Time\tArrival Time\tPriority\tWaiting Time\tTurnaround Time");
for (i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    total += tat[i];
    printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", p[i], bt[i], at[i], priority[i], wt[i], tat[i]);
}
avg_tat = (float) total / n;
printf("\n\nAverage Waiting Time = %f", avg_wt);
printf("\n\nAverage Turnaround Time = %f\n", avg_tat);
return 0;
}

```

## CONCLUSION:

In conclusion, this lab equipped us with a practical understanding of the trade-offs and nuances associated with different scheduling algorithms. The dynamic nature of real-world systems requires careful consideration of factors such as arrival times and priorities to design efficient and responsive systems. As we move forward in the study of operating systems, these fundamental concepts will serve as a solid foundation for tackling more advanced scheduling challenges.