OPERATING SYSTEMS
TUTORIAL ASSIGNMENT-01

Submitted By: AKHIL SINGH, *Enrollment No.* IIT2018198, *B.Tech. IT* 3rd Semester, *Section* -'B'

1: Write a brief description with an example of the following GCC command line options: -o, -E, -c, -S, -save-temps, -l, -D, -Werror.
Ans-
gcc -o : To compile and name executable output file name.
Example : gcc file.c -o file
file .c is compiled and executable by name of file is created.

gcc -E : gcc -E only invokes the preprocessor without compiling or using the linker.
Example : gcc -E file.c > file.i

gcc -c : With -c, the linker is not run.
Example : gcc -c file.c

gcc -S : Create assembler code (*.s)
Example : gcc -S file.c > file.s

gcc -save-temps : Store the normally temporary intermediate files(*.s, *.i, *.o) permanently.
Example : gcc -save-temps file.c

gcc -l : This is used to link to a particular library file with current file to be compiled.
Example : gcc file.c -o file -lm

gcc -D : GCC allows to define the value of macro with the -D option when invoked as compiler.
Example : gcc -DMY_MACRO file.c

gcc -Werror : Makes all warnings into errors.
Example : gcc -Werror file.c

2: Collect the following basic information about your machine using proc. How many CPU cores does the machine have? How much memory, and what fraction

of it is free? How many context switches has the system performed since bootup? How many processes has it forked since bootup?
Ans:
Number of Cores:
Use cat /proc/interrupts ->It gives details about interrupts caused in CPU core wise, it will list CPU cores as CPU0 CPU1 ……
To only list CPU cores use : cat /proc/interrupts | egrep -i ‘cpu’
Number of cores on my machine = 4.

Memory: cat /proc/meminfo
Output:

```
MemTotal:       3906364 kB
MemFree:         907316 kB
MemAvailable:    1856116 kB
```

Percentage free = (907316/3906364)*100 = 23.24%

Context Switches since Bootup: cat /proc/stat -> It gives number of context switches made in kernel since bootup.
Output : ctxt 7964136

Forks since Bootup: cat /proc/stat | grep processes-> Labeled as processes, the number of forks made since bootup is given.
Output: processes 4685

3: Every process consumes some resources (CPU, memory, disk or network bandwidth, and so on).When a process runs as fast as it can, one of these resources is fully utilized, limiting the maximum rate at which the process can make progress. Such a resource is called the bottleneck resource of a process. A process can be bottlenecked by different resources at different points of time, depending on the type of work it is doing. Run each of the four programs (cpu, cpu-print, disk, and disk1) separately, and identify what the bottleneck resource for each is (without reading the code). For example, you may monitor the utilization of various resources and see which ones are at the peak. Next, read through the code and justify how the bottleneck you identified is consistent with what the code does. For each of the programs, you must write down three things: the bottleneck resource, the reasoning that went into identifying the bottleneck, (e.g., the commands you ran, and the outputs you got),and a justification of the bottleneck from reading the code.

cpu.c:
Bottleneck resource - CPU usage.
Reason - Run command $top in a terminal, in another terminal run ./cpu
Output: CPU percentage usage -> 100%
Justification - This program consists of basic arithmetic operation running in an infinite loop which takes up the entire processing thus creating a bottleneck.

cpu-print.c:
Bottleneck resource - Disk-write usage.
Reason - Run command $iotop in a terminal, in another terminal run ./cpu-print
Output: Disk-Write tops out at about 1500KBps.
Justification - This program consists of printf invoking the write() system call running in an infinite loop which exhausts disk-write speed. Thus, creating a bottleneck for other resources and limiting speed of the process.

disk.c:
Bottleneck resource - Disk-read usage.
Reason - Run command $iotop in a terminal, in another terminal run ./cpu
Output: Disk-Read usage -> 45M/s
Justification - This program consists of fread() invoking read system call running in an infinite loop which exhausts disk read speed thus creating a bottleneck for other resources.

disk1.c:
Bottleneck resource - CPU usage.
Reason - Run command $top in a terminal, in another terminal run ./cpu
Output: CPU percentage usage -> 100%
Justification - This program consists of fread() invoking read system call running in an infinite loop which exhausts CPU usage creating a bottleneck for other resources.

4: Recall that every process runs in one of two modes at any time: user mode and kernel mode. It runs in user mode when it is executing instructions / code from the user. It executes in kernel mode when running code corresponding to system calls etc. Compare (qualitatively) the programs cpu and cpu-print in terms of the amount of time each spends in the user mode and kernel mode, using information from the proc file system. For examples, which programs spend more time in kernel mode than in user mode, and vice-versa? Read through their code and justify your observations.
Ans.

cpu.c : use $pidof ./cpu   [To get process id of ./cpu]
                  $cat /proc/<pid>/stat   [To get stat of process with given pid]
Look at column 14 for user-time and 15 for kernel-time in clock ticks.
cpu.c runs primarily in user-mode and almost none at all in kernel mode.
After about 5 sec it took approx. 2000 clock-ticks in user and 0 ticks in kernel mode.
After about 5 minutes it took approx. 70000 clock-ticks in user and 7 ticks in kernel mode. From reading the c-code it is clear that as it includes only basic arithmetic operation and no system call there is not much time spent in kernel mode and almost all the time is spent in user mode.

cpu-print.c :
cpu-print.c runs primarily in kernel-mode and a lot less in user mode.
Look at column 14 for user-time and 15 for kernel-time in clock ticks.
After about 5 sec it took approx. 450 clock-ticks in user and 1000 ticks in kernel mode.
From reading the c-code it is clear that as it includes printf() which makes use of write() syscall much of the time has to be spent in kernel mode and relatively lesser in user mode.

5: Recall that a running process can be interrupted for several reasons. When a process must stop running and give up the processor, it's CPU state and registers are stored, and the state of another process is loaded. A process is said to have experienced a context switch when this happens.Context switches are of two types: voluntary and involuntary. A process can voluntarily decide to give up the CPU and wait for some event, e.g., disk I/O. A process can be made to give up its CPU forcibly, e.g., when it has run on a processor for too long, and must give a chance to other processes sharing the CPU. The former is called a voluntary context switch, and the latter is called an involuntary context switch. Compare the programs cpu and disk in terms of the number of voluntary and involuntary context switches. Which program has mostly voluntary context switches, and which has mostly in voluntary context switches? Read through their code and justify your observations.
Ans:
cpu.c: Use $pidof ./cpu   [To get process id of ./cpu]
                 $cat /proc/<pid>/status
cpu.c causes primarily non-voluntary context switches. After 10 seconds about 0 voluntary context switches and 500 non-voluntary context switches had occurred.
Main reason for this occurrence is that cpu.c contains basic arithmetic operation code only which only requires processing usage and no time for I/O or other process that can

cause voluntary context switch, so in order to facilitate processing of other programs in queue non-voluntary context switch is a must.

disk.c: Use $pidof ./disk   [To get process id of ./disk]
    $cat /proc/<pid>/status
disk.c causes primarily voluntary context switches. After 10 seconds about 3000 voluntary context switches and 157 non-voluntary context switches had occurred. Main reason for this occurrence is that disk.c contains read call which is an I/O process that can cause voluntary context switch as cpu is free while input is being read. So this process involves a lot of voluntary context switching.

6: Create a library file and using the created library file to link and run a code. write functions of matrix operation with one function in one .c file. compile each file separately and then create a library from all these files. Then write one main.c file and use all these matrix operations by linking the main file with the newly created library.

Ans:

```
gcc -o mat_add.o -c mat_add.c
gcc -o mat_tr.o -c mat_tr.c
gcc -o mat_sub.o -c mat_sub.c
gcc -o mat_det.o -c mat_det.c
gcc -o mat_mul.o -c mat_mul.c
ar rcs libmylib.a mat_add.o mat_tr.o mat_sub.o mat_det.o mat_mul.o
gcc code.c -L. -lmylib -lm
```

funcdec.h->

```
#ifndef _FUNCDEC_H_
#define _FUNCDEC_H_

        void add(int n, int m, int matrix1[n][m], int matrix2[n][m], int result[n][m]);
        void transpose(int n, int m, int matrix1[n][m], int result[m][n]);
        void substract(int n, int m, int matrix1[n][m], int matrix2[n][m], int result[n][m]);
        int determinant(int n, int mat[n][n]);
        void multiply(int n, int m, int matrix[n][m], int y,int result[n][m]);
#endif
```

Code.c->

```
#include <stdio.h>
#include <math.h>
#include "funcdec.h"
```

```
void print(int m,int n,int mat[m][n]){
        for(int i=0;i<m;i++){
                for(int j=0;j<n;j++)
                        printf("%d ",mat[i][j]);
                printf("\n");
        }
        printf("\n");
}
int main(){
        int m=4,n=4;
        int mat1[4][4] = {{1, 2, 3, 1},
               {2, 3, 2, 4},
               {1, 5, 3, 3},
               {6, 1, 2, 2}};
   int mat2[4][4] = {{1, 4, 3, 0},
               {1, 1, 2, 4},
               {1, 3, 4, 7},
               {5, 1, 2, 2}};
   int rev[n][m],result[m][n];
        printf("Matrix 1 is:\n");print(m,n,mat1);
        printf("Matrix 2 is:\n");print(m,n,mat2);
        printf("Addition of Matrix 1 and 2 is :\n");
        add(n,m,mat1,mat2,result);print(m,n,result);
        printf("Substract of Matrix 1 and 2 is :\n");
        substract(n,m,mat1,mat2,result);print(m,n,result);
        printf("Multiplication of Matrix 1 with 6 is :\n");
        multiply(n,m,mat1,6,result);print(m,n,result);
        printf("Transpose of Matrix 2 is :\n");
        transpose(n,m,mat2,rev);print(n,m,rev);
        printf("Matrix 1's determinant is :%d\n",determinant(m,mat1));
        printf("Matrix 2's determinant is :%d\n",determinant(m,mat2));
return 0;
}
```

mat_add.c->

```
#include "funcdec.h"
void add(int n, int m, int matrix1[n][m], int matrix2[n][m], int result[n][m])
{
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<m;j++)
                {
                        result[i][j]=matrix1[i][j]+matrix2[i][j];
                }
        }
}
```

mat_sub.c->

```
#include "funcdec.h"

void substract(int n, int m, int matrix1[n][m], int matrix2[n][m], int result[n][m])
{
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<m;j++)
                {
                        result[i][j]=matrix1[i][j]-matrix2[i][j];
                }
        }
}
```

mat_det.c->

```
#include "funcdec.h"
#include<math.h>
int determinant(int n, int mat[n][n])
{
        int num1,num2,det=1,index,total=1;
        int temp[n+1];
        for(int i=0;i<n;i++)
        {
                index = i;
                while(mat[index][i]==0 && index<n)
        {
        index++;
        }
        if(index==n)
        {
                continue;
        }
        if(index!=i)
        {
                for(int j=0;j<n;j++)
                {
                        int t=mat[index][j];
                        mat[index][j]=mat[i][j];
                        mat[i][j]=t;
                }
                det=det*pow(-1,index-i);
        }
        for(int j=0;j<n;j++)
        {
                temp[j]=mat[i][j];
                }
                for(int j=i+1;j<n;j++)
                {
                        num1=temp[i];
                        num2=mat[j][i];
                        for(int k=0;k<n;k++)
                        {
                                mat[j][k]=(num1*mat[j][k])-(num2*temp[k]);
                        }
                                total=total*num1;
                        }
                }
                for(int i=0;i<n;i++)
                {
                        det=det*mat[i][i];
        }
        return(det/total);
}
```

mat_tr.c->

```
#include "funcdec.h"
void transpose(int n, int m, int matrix1[n][m], int result[m][n])
{
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<m;j++)
                {
                        result[i][j]=matrix1[j][i];
                }
        }
}
```