# Practical-1 (Introduction to Linux Commands)

---> Is practical me koi code nahi hai bhai

_ _

# Practical-2 (Shell Programming)

Q1. Write a shell script to write "Multiplication Tables"

---> Code:

```
#! /usr/bin/bash

echo "Enter a Number: "
read a

for((i=0; i<a; i++))
do
	b=$((a*i))
	echo "$a x $i = $b"
done
```

_ _

Q2. Write a shell script for a "Small Calculator"

---> Code:

```
#! /usr/bin/bash

echo "Enter 1st Number: "
read a

echo "Enter 2nd Number: "
read b
```

```
echo "Which operation do you wanna perform"
echo "1. Addition      2. Subtraction        3. Multiplication       4. Division         5. Exit Calculator
Interface"
read c

if [ $c -eq 1 ]
then
        let d=$a+$b
        echo "Sum of these two numbers is: $d"
fi

if [ $c -eq 2 ]
then
        echo " "
        echo "Do you wanna Subtract: "
        echo "1. 1st no. from 2nd no."
        echo "OR"
        echo "2. 2nd no. from 1st no."
        echo "Enter 1 or 2 as per your need."
        read e

        if [ $e -eq 1 ]
        then
                let d=$b-$a
                echo " "
                echo "When you subtract 1st no. from 2nd no. you get: $d"

        elif [ $e -eq 2 ]
        then
                let d=$a-$b
                echo " "
                echo "When you subtract 2nd no. from 1st no. you get: $d"
        fi
fi

if [ $c -eq 3 ]
then
        let d=$a*$b
        echo "Multiplication of these two numbers is: $d"
fi

if [ $c -eq 4 ]
then
        echo " "
        echo "Do you wanna Divide: "
        echo "1. 1st no. by 2nd no."
        echo "OR"
        echo "2. 2nd no. by 1st no."
        echo "Enter 1 or 2 number as per your need."
        read e

        if [ $e -eq 1 ]
        then
                echo " "
                echo "When you divide 1st no. by 2nd no. you get: "
                echo "scale=2; $a/$b" | bc

        elif [ $e -eq 2 ]
        then
                echo " "
                echo "When you divide 2nd no. by 1st no. you get: "
                echo "scale=2; $b/$a" | bc
        fi
fi
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q3. Write a shell script for "Diplaying prime numbers upto a given limit"

---> Code:

```
#! /usr/bin/bash

echo "Enter a Number: "
read a

for ((i=2; i<=a; i++))
do
        n=0
        for ((j=2; j<i; j++))
        do
                let b=$i%$j
                if [ $b -eq 0 ]
                then
                        n=1
                fi
        done

        if [ $n -eq 0 ]
        then
                echo $i
        fi
done
```

_ _

# Practical-3 (File Manipulation Using System Calls)

Q1. Using system calls copy first half of the content of a already existing file to a newly created file and then again copy the rest remaining second half of the content of that older file to a another newly created file

---> Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
        int a, b, c, n;
        char buff1[500];

        a = open("content.txt", O_RDONLY | O_CREAT, 0777);
        b = open("FirstHalf.txt", O_WRONLY | O_CREAT, 0777);
        c = open("SecondHalf.txt", O_WRONLY | O_CREAT, 0777);

        n = read(a, buff1, 500);

        read(a, buff1, n/2);
        write(b, buff1, n/2);

        lseek(a, n/2, SEEK_SET);
        read(a, buff1, n/2);
        write(c, buff1, n/2);

        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q2. Using system calls write a program which reads from console until user types '$' and the content which is written on the console before '$' copy that content to a newly created file

---> Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
        int a, b=0;
        char buff1[500];
        scanf("%[^\n]s", buff1);

        a = open("Output$.txt", O_WRONLY | O_CREAT, 0777);

        for(int i=0; i<500; i++){
                if(buff1[i] == '$'){
                        printf("You can not write after '$' Symbol \n");
                        break;
                }

                else{
                        b++;
                }
        }

        char buff2[b];

        for(int i=0; i<b; i++){
                buff2[i]=buff1[i];
        }

        write(a, buff2, b);
        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q3. Write a program using system call to read the contents of a file without using char array and display the contents on the console (Don't use any built in functions like sizeof() and strlen())

---> Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
	int a, b;
	a = open("Input.txt", O_RDONLY | O_CREAT, 0777);
	char buff[1];
	char *c = buff;

	while((b = read(a, c, 1))>0){
		write(1, c, 1);
	}

	return 0;
}
```

_ _

# Practical-4 (Directory Manipulation Using System Calls)

Q1. Write a program using directory system calls, make a directory on desktop and create a file inside the directory and list the contents of the directory

---> Code:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<dirent.h>

int main()
{
	int a = mkdir("My_Directory",0777);
```

```
        if(a!=-1)
        {
                printf("# You're directory has been created: \n");
                printf(" \n");
        }

        struct dirent *dptr;

        int fd1 = creat("My_Directory/testing_123.txt",0777);
        int fd2 = creat("My_Directory/noicee.txt",0777);
        int fd3 = creat("My_Directory/life_is_good.txt",0777);

        if(fd1!=-1 && fd2!=-1 && fd3!=-1)
        {
                printf("* file testing_123.txt is created.\n");
                printf("* noicee.txt is created.\n");
                printf("* life_is_good.txt is created.\n");
                printf(" \n");
        }

        DIR *dp = opendir("My_Directory");

        printf("@ List of files in created directory: \n");

        while(NULL!=(dptr = readdir(dp)))
        {
                printf("%s\n", dptr->d_name);
        }
        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q2. Write a program using directory and file manipulation system calls to copy the contents of one directory to a newly created directory

---> Code:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<dirent.h>
```

```
int main()
{
	struct dirent *dptr;

	int a = mkdir("My_Directory", 0777);
	int b = mkdir("Your_Directory", 0777);

	if(a==0 && b==0){
		printf("* Directories named 'My_Directory' and 'Your_Directory' has been
created Successfully. \n");
	}
	else{
		printf("* Either directories named 'My_Directory' & 'Your_Directory' already
exist or they were not able to create due to any Error. \n");
	}

	int fd1 = open("My_Directory/My_File.txt", O_CREAT|O_RDWR, 0777);

	printf("* File named 'My_File.txt' has been created Successfully in Directory named
'My_Directory'. \n");
	printf(" \n");

	char c[1000];

	printf("# Kuch to likh de yaar file me, copy karani hai file: \n");
	scanf("%[^\n]s", c);
	printf(" \n");

	int size=0;

	for(int i=0;i<100;i++)
	{
		if(c[i]=='\0'){
			break;
		}

		else{
			size++;
		}
	}

	write(fd1, c, size);
	printf("@ Given input text from user has been Successfully copied in file named
'My_File.txt'. \n");
	printf(" \n");

	DIR *dp = opendir("Your_Directory");
	int no_of_files=-2;

	while(NULL != (dptr=readdir(dp)))
	{
		no_of_files++;
	}

	printf("# No. of files in directory which is named 'Your_Directory' are: \n");
	printf("%d\n",no_of_files);
	printf(" \n");

	int fd2 = open("Your_Directory/My_File_Copy.txt", O_CREAT | O_RDWR, 0777);
	printf("* File named 'My_File_Copy.txt' has been created Successfully in Directory named 'Your_Directory'. \n");

	write(fd2, c, size);
	printf("* Content of File named 'My_File' in Directory named 'My_Directory' has been Successfully Copied to file named 'My_File_Copy.txt' which is in Directory named 'Your_Directory' \n");
	printf(" \n");

	// -----------------------------------------------------------------------------------------

	DIR *dp1 = opendir("Your_Directory");

	int new_no_of_files=-2;

	while(NULL != (dptr=readdir(dp1)))
	{
		new_no_of_files++;
	}

	printf("# Now no. of files in directory which is named 'Your_Directory' are: \n");
	printf("%d\n", new_no_of_files);
	printf(" \n");

	return 0;
}
```

_ _

# Practical-5 (Process Management Using System Calls)

Q1. Write a program using system calls for operation on process to stimulate n fork calls to create (2^n - 1) child processes

---> Code:

```
#include<stdio.h>              // for printf and scanf
#include<unistd.h>             // for fork() & getpid()
#include<sys/types.h>          // for fork() & getpid()

int main(){

        int n;

        printf("# Enter the no. of times you want to run the fork system call: ");
        scanf("%d", &n);

        for(int i=0; i<n; i++){
                pid_t r;
                r = fork();
                if(r==0){
                        printf("Current child process pid is %d \n", getpid());
                }
        }

        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q2. Write a program using system calls for operations on processes to create a heirarchy of processes P1 -> P2 -> P3, also print the id and parent id for each process

---> Code:

```
#include<stdio.h>             //  for printf and scanf
#include<unistd.h>            //  for fork(), getpid() & getppid()
#include<sys/types.h>         //  for fork(), getpid() & getppid()
#include <stdlib.h>           //  for exit()

int main()
{
        printf("Parent PID : %d \n", (int) getpid());
```

```
    pid_t pid = fork();

    if(pid == 0)
    {
        printf("Child 1 PID : %d Parent PID : %d\n", (int) getpid(), (int) getppid());
        pid_t pid_1 = fork();
        if(pid_1 == 0)
        {
            printf("Child 2 PID : %d Parent PID (Child 1) : %d \n", (int) getpid(),
(int) getppid());
            exit(0);
        }
        else
        {
            exit(0);
        }
    }
    else
    {
        exit(0);
    }

    return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q3. Write a program using system calls for operations on processes to create a heirarchy of processes: P3 <- P2 <- P1 -> P4 -> P5, also stimulate process P4 as orphan and P5 as zombie

---> Code:

```
#include<stdio.h>               //  for printf and scanf
#include<unistd.h>              //  for fork(), getpid(), sleep() & getppid()
#include<sys/types.h>           //  for fork(), getpid() & getppid()
#include<stdlib.h>              //  for exit()


// In Orphan Process, child process P4 goes to sleep and whenever the sleep time period is
completed and P4 comes back for execution it's parent has already completed it's execution.
So, it will get a garbage parent PID, whenever it wants to access it's parents PID as it is a
orphan process.
```

// In Zombie Process, the parent process P4 goes to sleep and the child process P5 executes before P4 and leaves, and then after the completion of sleep time period process P4 executes, so here child process P5 will be called zombie process beacuse it has executed before it's parent's execution, and for P4 (parent) the process P5 is still visible in it's table but it has already completed it's execution.

```
int main()
{
        printf("P1 PID : %d \n", (int) getpid());
        pid_t pid = fork();

        if(pid == 0)
        {
                printf("P4 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
                printf("Child process P4 is sleeping \n");
                pid_t pid_1 = fork();
                sleep(5);
                if(pid_1 == 0)
                {
                        printf("P5 PID : %d P4 PID : %d \n", (int) getpid(), (int) getppid());
                        printf("Zombie process P5's PID : %d \n", (int) getpid());
                }
                else{
                        printf("Orphan child process P4's PID : %d \n", (int) getpid());
                        printf("P4's New Parent PID : %d \n", (int) getppid());
                }
        }
        else
        {
                pid = fork();
                if(pid == 0)
                {
                        printf("P2 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
                        pid_t pid_1 = fork();
                        if(pid_1 == 0)
                        {
                                printf("P3 PID : %d P2 PID : %d \n", (int) getpid(), (int)
getppid());
                                exit(0);
                        }
                        else
                        {
                                exit(0);
                        }
                }
```

```
		else
		{
			exit(0);
		}
	}

	return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q4. Write a program using system calls for operations on processes to create a heirarchy of processes: P4 <- P3 <- P2 <- P1 -> P5 -> P6 -> P7, also stimulate process P4 as an orphan process and P7 as zombie process

---> Code:

```
#include<stdio.h>              //  for printf and scanf
#include<unistd.h>              //  for fork(), getpid(), sleep() & getppid()
#include<sys/types.h>            //  for fork(), getpid() & getppid()
#include<stdlib.h>              //  for exit()

int main()
{
	printf("P1 PID : %d \n", (int) getpid());

	pid_t pid = fork();

	if(pid == 0)
	{
		printf("P5 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getppid());
		pid_t pid_1 = fork();
		if(pid_1 == 0)
		{
			printf("P6 PID : %d Parent P5 PID : %d \n", (int) getpid(), (int)
getppid());

			pid_t pid_2 = fork();
			sleep(5);

			if(pid_2 == 0)
			{
				printf("Zombie process P7's PID: %d \n", (int) getpid());
				printf("Parent P6 PID : %d \n", (int) getppid());
			}
```

```
                else
                {
                    exit(0);
                }
            }
            else
            {
                exit(0);
            }
        }

        else
        {
            pid = fork();
            if(pid == 0)
            {
                printf("P2 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int)
getppid());
                pid_t pid_1 = fork();
                if(pid_1 == 0)
                {
                    printf("P3 PID : %d Parent P2 PID : %d \n", (int) getpid(), (int)
getppid());

                    pid_t pid_2 = fork();
                    if(pid_2 == 0)
                    {
                        sleep(3);
                    }
                    else
                    {
                        printf("Orphan child process P4's PID : %d \n", (int)
pid_2);
                        printf("P4's New Parent PID : %d \n", (int) getppid());
                    }
                }
                else
                {
                    exit(0);
                }
            }
        }

        return 0;
}
```

_ _

# Practical-6 (Creation of Multithreaded Processes using Pthread Library)

Q1. Write a program using pthread to concatenate the strings, where multiple strings are passed to thread function

---> Code:

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>


char str1[100], str2[100];
char result[1000];


void *concatenatestrings(){
	strcat(result, str1);
	strcat(result, str2);
	pthread_exit(NULL);
}

int main(){
	pthread_t thread;
	printf("* Enter the first string: ");
	scanf("%s", str1);
	printf("* Enter the second string: ");
	scanf("%s", str2);

	pthread_create(&thread, NULL, concatenatestrings, NULL);
	pthread_join(thread, NULL);

	printf("@ Final result is: %s \n", result);
	return 0;
}
```

_ _

Q2. Write a program using pthread to find the length of string, where strings are passed to thread function

---> Code:

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>

char length1[100];
int length=0;

void *lengthstr(){
        length=strlen(length1);
        pthread_exit(NULL);
}


int main(){
        pthread_t thread;
        printf("* Enter the String: ");
        scanf("%[^\n]s", length1);

        pthread_create(&thread, NULL, lengthstr, NULL);
        pthread_join(thread, NULL);
        printf("* Total length of string is: %d \n", length);
        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q3. Write a program that performs statistical operations of calculating the average, maximum & minimum for a set of numbers. Create three threads where each performs their respective operations.

---> Code:

```
#include<stdio.h>
#include<pthread.h>

int arr[10] = {99, 22, 00, 88, 11, 102, 33, 66, 44, 55};

void *sort(){
	for(int i=0; i<10; i++){
		for(int j=0; j<10; j++){
			if(arr[i] < arr[j]){
				int temp = arr[i];
				arr[i] = arr[j];
				arr[j] = temp;
			}
		}
	}
}


void *min(){
	int min = arr[0];
	printf("* Minimum element is = %d\n", min);
	pthread_exit(NULL);
}


void *max(){
	int max = arr[9];
	printf("* Maximum element is = %d \n", max);
	pthread_exit(NULL);
}


void *avg(){
	int sum=0;
	for(int i=0;i<10;i++)
	{
		sum = sum + arr[i];
	}
	sum = sum/10;

	printf("* The average of the elements = %d \n", sum);
	printf("\n");
	pthread_exit(NULL);
}


int main(){

	printf("\n");

	/*
	printf("Enter 10 elements in the array: ");
	for(int i=0; i<10; i++)
	{
```

```
                scanf("%d", &arr[i]);
        }
        printf("\n");
        */

        printf("# Initial input array is: ");
        for(int i=0; i<10; i++){
                printf("%d ", arr[i]);
        }
        printf("\n");


        pthread_t sort_thread, max_thread, min_thread, avg_thread;

        pthread_create(&sort_thread, NULL, sort, NULL);
        pthread_join(sort_thread, NULL);

        pthread_create(&max_thread, NULL, max, NULL);
        pthread_join(max_thread, NULL);

        pthread_create(&min_thread, NULL, min, NULL);
        pthread_join(min_thread, NULL);

        pthread_create(&avg_thread, NULL, avg, NULL);
        pthread_join(avg_thread, NULL);
        return 0;
}
```

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q4. Write a multithreaded program where an array of integers is passed globally and is divided into two smaller lits and given as input to two threads. The thread will sort their half of the list and will pass the sorted list to a third thread which merges and sorts the list. The final sorted list is printed by the parent thread.

---> Code:

```
#include<stdio.h>
#include<pthread.h>

int arr[10] = {99, 22, 00, 88, 11, 100, 33, 66, 44, 55};


int arr_first_half[5], arr_second_half[5], final_arr[10];
```

```
void *final_merge_sort(){

        for(int i=0; i<5; i++){
                final_arr[i] = arr_first_half[i];
                final_arr[i+5] = arr_second_half[i];
        }

        printf("# Merged array is: ");
        for(int i=0; i<10; i++){
                printf("%d ", final_arr[i]);
        }
        printf("\n");

        for(int i=0; i<10; i++){
                for(int j=0; j<10; j++){
                        if(final_arr[i] < final_arr[j]){
                                int temp = final_arr[i];
                                final_arr[i] = final_arr[j];
                                final_arr[j] = temp;
                        }
                }
        }

        printf("@ Final Merged & Sorted array is: ");
        for(int i=0; i<10; i++){
                printf("%d ", final_arr[i]);
        }
        printf("\n");
        printf("\n");

        pthread_exit(NULL);
}


void *individual_sort(){
        for(int i=0; i<5; i++){
                for(int j=0; j<5; j++){
                        if(arr_first_half[i] < arr_first_half[j]){
                                int temp = arr_first_half[i];
                                arr_first_half[i] = arr_first_half[j];
                                arr_first_half[j] = temp;
                        }
                        if(arr_second_half[i] < arr_second_half[j]){
                                int temp = arr_second_half[i];
                                arr_second_half[i] = arr_second_half[j];
                                arr_second_half[j] = temp;
```

```
                    }
                }
            }

            pthread_exit(NULL);
}


int main()
{
            printf("\n");

            /*
            printf("Enter 10 elements in the array: ");
            for(int i=0; i<10; i++)
            {
                        scanf("%d", &arr[i]);
            }
            printf("\n");
            */

            printf("# Initial input array is: ");
            for(int i=0; i<10; i++){
                        printf("%d ", arr[i]);
            }
            printf("\n");


            for(int i=0; i<5; i++){
                        arr_first_half[i] = arr[i];
                        arr_second_half[i] = arr[i+5];
            }



            pthread_t parent_thread;
            pthread_create(&parent_thread, NULL, individual_sort, NULL);
            pthread_join(parent_thread, NULL);

            printf("* First half sorted array is: ");
            for(int i=0; i<5; i++){
                        printf("%d ", arr_first_half[i]);
            }
            printf("\n");


            printf("* Second half sorted array is: ");
            for(int i=0; i<5; i++){
                        printf("%d ", arr_second_half[i]);
            }
            printf("\n");

            pthread_create(&parent_thread, NULL, final_merge_sort, NULL);
            pthread_join(parent_thread, NULL);

            return 0;
}
```

_ _

# Practical-7th (Process Synchronization Using Semaphores/Mutex):

* Race Around Condition:

---> Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int NUM_THREADS = 2;
int shared = 0;

void *thread_func(void *arg) {
        int id = *(int *) arg;
        int local = 0;

        for (int i = 0; i < 1000000; i++) {
        local = shared;
        local++;
        shared = local;
        }

        printf("Thread %d: shared = %d\n", id, shared);

        pthread_exit(NULL);
}

int main() {
        pthread_t threads[NUM_THREADS];
        int thread_ids[NUM_THREADS];

        for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, thread_func, (void *) &thread_ids[i]);
        }

        for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
        }

        printf("Final value of shared = %d\n", shared);

        return 0;
}
```

_ _

* Race Around Condition Solved Using Mutex:

---> Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int NUM_THREADS = 2;
int shared = 0;

pthread_mutex_t mutex;

void *thread_func(void *arg) {
   int id = *(int *) arg;
   int local = 0;
   pthread_mutex_lock(&mutex);
   for (int i = 0; i < 1000000; i++) {
         local = shared;
         local++;
         shared = local;
   }
   printf("Thread %d: shared = %d\n", id, shared);
   pthread_mutex_unlock(&mutex);
   pthread_exit(NULL);
}

int main() {
   pthread_t threads[NUM_THREADS];
   int thread_ids[NUM_THREADS];
   pthread_mutex_init(&mutex,NULL);
   for (int i = 0; i < NUM_THREADS; i++) {
         thread_ids[i] = i;
         pthread_create(&threads[i], NULL, thread_func, (void *) &thread_ids[i]);
   }

   for (int i = 0; i < NUM_THREADS; i++) {
         pthread_join(threads[i], NULL);
   }

   printf("Final value of shared = %d\n", shared);

   return 0;
}
```

_ _

* Race Around Condition Solved Using Semaphore:

---> Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>


int NUM_THREADS = 2;
int shared = 0;

sem_t semaphore;

void *thread_func(void *arg) {
   int id = *(int *) arg;
   int local = 0;
   sem_wait(&semaphore);
   for (int i = 0; i < 1000000; i++) {
         local = shared;
         local++;
```

```
            shared = local;
    }
    printf("Thread %d: shared = %d\n", id, shared);
    sem_post(&semaphore);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];
    sem_init(&semaphore,0,1);
    for (int i = 0; i < NUM_THREADS; i++) {
            thread_ids[i] = i;
            pthread_create(&threads[i], NULL, thread_func, (void *) &thread_ids[i]);
    }

    for (int i = 0; i < NUM_THREADS; i++) {
            pthread_join(threads[i], NULL);
    }

    printf("Final value of shared = %d\n", shared);
    sem_destroy(&semaphore);
    return 0;
}
```

_ _

* Producer & Consumer Using Mutex

---> Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>


int BUFFER_SIZE = 5;
int buffer[5];
int count = 0;
int last_consumed_index = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_producer = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_consumer = PTHREAD_COND_INITIALIZER;
```

```
void* producer(void* arg) {
    int item;
    int iterations = 0;
    while (iterations < 10) { // exit after 10 iterations
        item = rand() % 100; // generate a random item
        pthread_mutex_lock(&mutex);
        if (count == BUFFER_SIZE) {
            pthread_cond_wait(&cond_producer, &mutex);
        }
        if (count == 0) {
            last_consumed_index = 0; // reset last consumed index if buffer is empty
        }
        buffer[last_consumed_index++] = item;
        printf("Produced item: %d\n", item);
        count++;
        if (count == 1) {
            pthread_cond_signal(&cond_consumer);
        }
        pthread_mutex_unlock(&mutex);
        iterations++;
    }
    return NULL;
}

void* consumer(void* arg) {
    int item;
    int iterations = 0;
    while (iterations < 10) { // exit after 10 iterations
        pthread_mutex_lock(&mutex);
        if (count == 0) {
            pthread_cond_wait(&cond_consumer, &mutex);
        }
        item = buffer[--last_consumed_index];
        printf("Consumed item: %d\n", item);
        count--;
        if (count == BUFFER_SIZE - 1) {
            pthread_cond_signal(&cond_producer);
        }
        pthread_mutex_unlock(&mutex);
        iterations++;
    }
    return NULL;
}

int main() {
    pthread_t producer_thread, consumer_thread;
    srand(time(NULL)); // initialize the random seed
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    return 0;
}
```

_ _

* Reader Writer Problem using Semaphore:

---> Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int NUM_READERS = 3;
int NUM_WRITERS = 2;
int MAX_ATTEMPTS = 5;

// Shared data
int shared_data = 0;
int num_readers = 0;

// Semaphores
sem_t mutex;
sem_t wrt;

// Reader function
void *reader(void *arg) {
    int id = *(int*)arg;
    int attempts = 0;

    while (attempts < MAX_ATTEMPTS) {
        // Entry section
        sem_wait(&mutex);
        num_readers++;
        if (num_readers == 1) {
            sem_wait(&wrt);
        }
        sem_post(&mutex);

        // Critical section
        printf("Reader %d read shared_data as %d\n", id, shared_data);

        // Exit section
        sem_wait(&mutex);
        num_readers--;
        if (num_readers == 0) {
            sem_post(&wrt);
        }
        sem_post(&mutex);

        attempts++;
    }

    pthread_exit(NULL);
}

// Writer function
void *writer(void *arg) {
    int id = *(int*)arg;
    int attempts = 0;

    while (attempts < MAX_ATTEMPTS) {
        // Entry section
        sem_wait(&wrt);

        // Critical section
        shared_data++;
        printf("Writer %d wrote shared_data as %d\n", id, shared_data);

        // Exit section
        sem_post(&wrt);

        attempts++;
    }

    pthread_exit(NULL);
}

int main() {
    // Initialize semaphores
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);
    // Create reader threads
    pthread_t reader_threads[NUM_READERS];
    int reader_ids[NUM_READERS];
    for (int i = 0; i < NUM_READERS; i++) {
        reader_ids[i] = i;
        pthread_create(&reader_threads[i], NULL, reader, &reader_ids[i]);
    }

    // Create writer threads
    pthread_t writer_threads[NUM_WRITERS];
    int writer_ids[NUM_WRITERS];
    for (int i = 0; i < NUM_WRITERS; i++) {
        writer_ids[i] = i;
        pthread_create(&writer_threads[i], NULL, writer, &writer_ids[i]);
    }

    // Wait for threads to finish
    for (int i = 0; i < NUM_READERS; i++) {
        pthread_join(reader_threads[i], NULL);
    }
    for (int i = 0; i < NUM_WRITERS; i++) {
        pthread_join(writer_threads[i], NULL);
    }

    // Destroy semaphores
    sem_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

_ _

# Practical-8th (Inter Process Communication Using Pipes/Shared Memory/RCP)

* Establish Interprocess communication (IPC) between Parent and child process using unnamed pipe.

---> Code:

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


int main() {
        int pipefd[2]; // file descriptors for the pipe
        char buffer[25];
        pid_t pid;

        if (pipe(pipefd) == -1) { // create the pipe
        printf("Pipe failed\n");
        return 1;
        }

        pid = fork(); // create a child process

        if (pid < 0) { // fork failed
        printf("Fork failed\n");
        return 1;
        }

        if (pid > 0) { // parent process
        close(pipefd[0]); // close the read end of the pipe
        printf("Parent process writing to pipe...\n");
        write(pipefd[1], "Hello, child process!", 22);
        close(pipefd[1]); // close the write end of the pipe
        }
        else { // child process
        close(pipefd[1]); // close the write end of the pipe
        printf("Child process reading from pipe...\n");
        read(pipefd[0], buffer, 25);
        printf("Child process received: %s\n", buffer);
        close(pipefd[0]); // close the read end of the pipe
        }

        return 0;
}
```

_ _

* Establish Interprocess communication (IPC) between Parent and child process using named pipe.

---> Code:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(){

        int pid, fd1, fd2;
        char buffer[20];

        mkfifo("my_Pipe", 0666);

        pid=fork();
        if(pid > 0){
                //Parent Section
                fd1=open("my_Pipe", O_WRONLY);
                write(fd1, "Hello Child Process\n",20);
        }
        if(pid==0){
                //Child section
                fd2=open("my_Pipe", O_RDONLY);
                read(fd2, buffer, 20);
                printf("%s", buffer);
        }

        return 0;
}
```

PEACE TE OUT
###