CPUBenchmarkF.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

//This function is for performing floating point operation to calculate the CPU performance

void* threadOperation(void *arg)
{
	int i;
	for(i=0;i<1000000000;++i)
		{
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
			5.6 + 2.6;
```

```
                        5.6 + 2.6;
                        5.6 + 2.6;
                }
return NULL;
}

// Return the time difference between two clocks

double timediff(clock_t t1, clock_t t2) {
    double elapsed;
    elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC;
    return elapsed;
}


int main(int argc, char *argv[]) {

        FILE *fp;
        fp=fopen("log_for_Float.txt", "a+");

        int no_of_threads=atoi(argv[1]);
        pthread_t tid[no_of_threads];
        int i,j,join_ret,thread_ret;

        clock_t t1,t2;
        clock_t totaltime;
        double flops_Count;
        float gflops_Count,flops_time;

        t1=clock();

        // Creating different number of threads depends on command line argument
        for(i=0;i<no_of_threads;i++)
        {
                pthread_create(&(tid[i]),NULL,threadOperation,NULL);
        }
        // Join the thread so that main program execution will be in waiting state
        for(j=0;j<no_of_threads;j++)
        {
                pthread_join(tid[j],NULL);
        }

        t2=clock();
        double elapsed = timediff(t1,t2);
        //printf("elapsed: %lf sec\n", elapsed);
        flops_Count = no_of_threads*20*(1000000000/elapsed);
        gflops_Count = flops_Count / 1000000000;
        fprintf(fp, "GFLOPS: %f\n",gflops_Count);
```

```
    fclose(fp);
    return 0;


}
```

CPUBenchmarkI.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

//This function is for performing Intergers operation to calculate the CPU performance
void* threadOperation(void *arg)
{

        int i;
        for(i=0;i<1000000000;++i)
                {
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
```

```
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;
                        4+2;

                }
return NULL;
}

// Return the time difference between two clocks

double timediff(clock_t t1, clock_t t2) {
    double elapsed;
    elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC;
    return elapsed;
}

int main(int argc, char *argv[]) {

        FILE *fp;
        fp=fopen("log_for_Integer.txt", "a+");

        int no_of_threads=atoi(argv[1]);
        pthread_t tid[no_of_threads];
        int i,j,join_ret,thread_ret;

        clock_t t1,t2;
        double iops_Count,elapsed;
        float giops_Count;

        t1=clock();

        // Creating different number of threads depends on command line argument
        for(i=0;i<no_of_threads;i++)
        {
                pthread_create(&(tid[i]),NULL,threadOperation,NULL);
        }

        // Join the thread so that main program execution will be in waiting state
        for(j=0;j<no_of_threads;j++)
        {
                pthread_join(tid[j],NULL);
        }
        t2=clock();
```

```
        elapsed= timediff(t1,t2);
        //printf("elapsed: %ld ms\n", elapsed);

        iops_Count = no_of_threads*20*(1000000000/ elapsed);
        giops_Count = iops_Count / 1000000000;
        fprintf(fp, "GIOPS: %f\n",giops_Count);
        //printf("\nGFLOPS = %f\n", giops_Count);

   fclose(fp);
   return 0;


}
```

BenchmarkF.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

//This array will hold per second executed  number of instructions so that we can keep the log of their
values

long catlog[4];

//This function is for performing Floating operation to calculate the CPU performance

void* threadOperation(void *arg)
{
        int count= * (int *)arg;
```

```
	int i;

	//for(i=0;i<100000;i++)
	while(1)
		{
		int addition = 4.4+2.4 + 4.6+2.4 + 4.4+2.4 + 4.4+2.4 + 4.7+2.2 + 4.6+2.2+ 4.2+2.2 +
	4.5+2.5 + 4.7+2.3 +    4.9+2.5 + 5.4+2.4 +    7.4+2.3 + 8.4+2.2 + 3.4+2.1 +5.5+2.2 + 6.1+2.3 +
9.4+2.3 + 4.4+2.3 + 4.4+2.8 + 4.3+2.6 + 3.4+2.4;
		catlog[count]=catlog[count]+20;
		}
return NULL;
}

// Return the time difference between two clocks

double timediff(clock_t t1, clock_t t2) {
   double elapsed;
   elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC;
   return elapsed;
}

int main(int argc, char *argv[]) {

	FILE *fp;
	fp=fopen("log_for_Experiment_Floating.txt", "a+");

	int no_of_threads=atoi(argv[1]);
	pthread_t tid[no_of_threads];
	int i,j,k,l,join_ret,thread_ret;
	int p[4];
	l=0;

	// Creating different number of threads depends on command line argument

	for(k=0;k<4;k++)
	{
		catlog[k]=0;
		p[k]=k;
	}

	clock_t t1,t2,t3,t4;
	clock_t totaltime;
	//double flops_time=0;
	double flops_Count;
	float gflops_Count,flops_time;

	t1=clock();
```

```
	for(i=0;i<no_of_threads;i++)
	{
		pthread_create(&(tid[i]),NULL,threadOperation,&p[i]);
	}
	fprintf(fp,"----------------------------------------------------------\n");

	// Here will calculate the per seconds number of operations performed by all threads combined
	// and will track this till 600 seoinds and after that will kill the running thread processess.

	while(1)
	{
		t2=clock();
		if(((t2-t1)/CLOCKS_PER_SEC)>=1)
		{
			t1=clock();
			l++;
			catlog[0]=catlog[0]+catlog[1]+catlog[2]+catlog[3];

			fprintf(fp, "%ld\n",catlog[0] / 1000000000);
			catlog[0]=0;
			catlog[1]=0;
			catlog[2]=0;
			catlog[3]=0;
		}
		if(l==600)
		{
			for(i=0;i<no_of_threads;i++)
				{
					pthread_kill(&(tid[i]),1);
				}
			break;
		}
	}

   fclose(fp);
   return 0;


}
```

BenchmarkI.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
```

```
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

//This array will hold per second executed  number of instructions so that we can keep the log of their
values

long catlog[4];

//This function is for performing Intergers operation to calculate the CPU performance

void* threadOperation(void *arg)
{
        int count= * (int *)arg;

        int i;

        //for(i=0;i<100000;i++)
        while(1)
                {
                int addition =
4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+2+4+
2+4+2;
                catlog[count]=catlog[count]+20;
                }
return NULL;
}

// Return the time difference between two clocks

double timediff(clock_t t1, clock_t t2) {
   double elapsed;
   elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC;
   return elapsed;
}

int main(int argc, char *argv[]) {
```

```
FILE *fp;
fp=fopen("log_for_Experiment_Integer.txt", "a+");

int no_of_threads=atoi(argv[1]);
pthread_t tid[no_of_threads];
int i,j,k,l,join_ret,thread_ret;
int p[4];
l=0;

// Creating different number of threads depends on command line argument

for(k=0;k<4;k++)
{
        catlog[k]=0;
        p[k]=k;
}

clock_t t1,t2,t3,t4;
clock_t totaltime;
//double flops_time=0;
double flops_Count;
float gflops_Count,flops_time;

t1=clock();

for(i=0;i<no_of_threads;i++)
{
        pthread_create(&(tid[i]),NULL,threadOperation,&p[i]);
}
fprintf(fp,"-----------------------------------------------------------\n");

// Here will calculate the per seconds number of operations performed by all threads combined
// and will track this till 600 seoinds and after that will kill the running thread processess.

while(1)
{
        t2=clock();
        if((t2-t1)/CLOCKS_PER_SEC>=1)
        {
                t1=clock();
                l++;

                catlog[0]=catlog[0]+catlog[1]+catlog[2]+catlog[3];

                fprintf(fp, "%ld\n",catlog[0] / 1000000000);
                catlog[0]=0;
                catlog[1]=0;
                catlog[2]=0;
                catlog[3]=0;
```

```
                }
                if(l==600)
                {
                        for(i=0;i<no_of_threads;i++)
                                {
                                        pthread_kill(&(tid[i]),1);
                                }
                        break;
                }
        }

   fclose(fp);
   return 0;


}
```

MemoryBenchmark.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

char buffer[1024*1024];
//char * source_buffer;
char * destination_buffer;
char * source_buffer;
```

```
// this is buffered space we are using for reading and writing purpose form Disk and to disk

// Return the time difference between two clocks
long timediff(clock_t t1, clock_t t2) {
    long elapsed;
    elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC * 1000;
    return elapsed;
}

// this is to Read+Write a byte data sequencially from Memory
void* readSeqByte(void *arg)
{
        int i;
        int source_buffer_index,destination_buffer_index;
        source_buffer_index=0;
        destination_buffer_index=0;
/*      char * destination_buffer = (char *) malloc(1024*1024*100);
        char * source_buffer= (char *) malloc(1024*1024*100);*/

        for(i=0;i<100000000;i++) // 100mb data
        {
                memcpy(&destination_buffer[destination_buffer_index],
&source_buffer[source_buffer_index], 1);
                source_buffer_index= source_buffer_index +1;
                destination_buffer_index=destination_buffer_index+1;
                //source_buffer=source_buffer+1;
                //destination_buffer=destination_buffer+1;
        }
        return NULL;
}

// this is to Read+Write a Kilo byte data sequencially from Memory
void* readSeqKByte(void *arg)
{
        int i;
        int source_buffer_index,destination_buffer_index;
/*      char * destination_buffer = (char *) malloc(1024*1024*100);
        char * source_buffer= (char *) malloc(1024*1024*100);*/
        source_buffer_index=0;
        destination_buffer_index=0;
        for(i=0;i<100000;i++) // 100mb data
        {
                memcpy(&destination_buffer[destination_buffer_index],
&source_buffer[source_buffer_index], 1024);

                source_buffer_index= source_buffer_index +1024;
                destination_buffer_index=destination_buffer_index+1024;
                //source_buffer=source_buffer+1024;
```

```
                //destination_buffer=destination_buffer+1024;
        }
        return NULL;
}

// this is to Read+Write a Mega byte data sequencially from Memory
void* readSeqMByte(void *arg)
{
        int i;
        int source_buffer_index,destination_buffer_index;
        source_buffer_index=0;
                destination_buffer_index=0;
/*      char * destination_buffer = (char *) malloc(1024*1024*100);
        char * source_buffer= (char *) malloc(1024*1024*100);*/

        //printf("\nSizeof Source buffer = %lf\n",sizeof(source_buffer));

        for(i=0;i<100;i++) // 100mb data
        {
                memcpy(&destination_buffer[destination_buffer_index],
&source_buffer[source_buffer_index], (1024*1024));
                source_buffer_index= source_buffer_index +(1024*1024);
                destination_buffer_index=destination_buffer_index+(1024*1024);
                /*
                source_buffer=source_buffer+(1024*1024);
                destination_buffer=destination_buffer+(1024*1024);*/
        }
        return NULL;
}

// this is to Read+Write a byte data Randomly from Memory
void* readRanByte(void *arg)
{
        int i,randloc;
        char * destination_buffer = (char *) malloc(1024*1024*100);
        char * source_buffer= (char *) malloc(1024*1024*100);

        for(i=0;i<100000000;i++) // 100mb data
        {
                randloc= rand() % (1024*1024*100);
                memcpy(&destination_buffer[randloc], &source_buffer[randloc], 1);
        }
        return NULL;
}

// this is to Read+Write a Kilo byte data Randomly from Memory
void* readRanKByte(void *arg)
{
        int i,randloc;
```

```
/*	char * destination_buffer = (char *) malloc(1024*1024*100);
	char * source_buffer= (char *) malloc(1024*1024*100);*/

	for(i=0;i<100000;i++) // 100mb data
	{
		randloc= (rand() % ((1024*1024*100)-1024));
		memcpy(&destination_buffer[randloc], &source_buffer[randloc], 1024);
	}
	return NULL;
}

// this is to Read+Write a Mega byte data Randomly from Memory
void* readRanMByte(void *arg)
{
	int i,randloc;

/*	char * destination_buffer = (char *) malloc(1024*1024*100);
	char * source_buffer= (char *) malloc(1024*1024*100);*/

	for(i=0;i<100;i++) // 100mb data
	{
		randloc= (rand() % ((1024*1024*100)-(1024*1024)));
		memcpy(&destination_buffer[randloc], &source_buffer[randloc], (1024*1024));
	}
	return NULL;
}



int main(int argc, char *argv[]) {

	FILE *log;
	log=fopen("Memory_log.txt", "a+");

	//source_buffer= (char *) malloc(1024*1024*100);
	destination_buffer = (char *) malloc(1024*1024*100);
	source_buffer= (char *) malloc(1024*1024*100);

	int i,j,cnt=0;
	size_t size = 1;

	int number_of_operation=1;
	double latency,totaldata;
	double elapsed,throughput;
	int no_of_threads = atoi(argv[1]);
	pthread_t tid[no_of_threads];

	// Initialize the buffer so that we can write it in memory
	//printf("\nInitialize the buffer\n");
```

```
for(i=0;i<(1024*1024);i++)
{
	buffer[i]='k';
}

//--------------ReadWriteSeqByte----------------------//

//printf("\nReadWriteSeqByte\n");
clock_t t1,t2;
t1=clock();
for (i = 0; i < no_of_threads; i++)
{
	pthread_create(&(tid[i]),NULL,readSeqByte,NULL);
}
for (j = 0; j < no_of_threads; j++)
{
	pthread_join(tid[j],NULL);
}
t2=clock();
elapsed = timediff(t1,t2);
//printf("ReadWriteSeqByte operation %lf  milliseconds\n", elapsed);
fprintf(log, "ReadWriteSeqByte operation %lf  milliseconds\n", elapsed);

//printf("ReadSeqByte: %lf ms\n", elapsed);
totaldata = (double) (no_of_threads * 100000000 * 1) / 1000000;
latency = (double) (elapsed / totaldata);
fprintf(log, "Latency: time per operation %lf  \n", latency);

//printf("Throughput: operation per milliseconds %lf  \n", throughput);
elapsed = elapsed / 1000;
throughput = (double) (totaldata / elapsed);
fprintf(log, "Throughput: operations(mb) per seconds %lf  \n", throughput);


//--------------ReadWriteSeqKByte----------------------//
//printf("\nReadWriteSeqKByte\n");
clock_t t3,t4;
t3=clock();
for (i = 0; i < no_of_threads; i++)
{
	pthread_create(&(tid[i]),NULL,readSeqKByte,NULL);
}
for (j = 0; j < no_of_threads; j++)
{
	pthread_join(tid[j],NULL);
}
t4=clock();
elapsed = timediff(t3,t4);
//printf("ReadWriteSeqKByte operation %lf  milliseconds\n", elapsed);
```

```
fprintf(log, "ReadWriteSeqKByte operation %lf  milliseconds\n", elapsed);

totaldata= (double)(no_of_threads*100000*1024)/1000000;
	latency= (double)(elapsed/totaldata);
	fprintf(log, "Latency: time per operation %lf  \n", latency);

	//printf("ReadSeqKByte: %lf ms\n", elapsed);
	elapsed=elapsed/1000;
	throughput= (double)(totaldata/elapsed);
	fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);

//--------------ReadWriteSeqMByte----------------------//

//printf("\nReadWriteSeqMByte\n");
clock_t t5,t6;
t5=clock();
for (i = 0; i < no_of_threads; i++)
{
	pthread_create(&(tid[i]),NULL,readSeqMByte,NULL);
}
for (j = 0; j < no_of_threads; j++)
{
	pthread_join(tid[j],NULL);
}
t6=clock();
elapsed = timediff(t5,t6);
//printf("ReadWriteSeqMByte operation %lf  milliseconds\n", elapsed);
fprintf(log, "ReadWriteSeqMByte operation %lf  milliseconds\n", elapsed);
totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
	latency= (double)(elapsed/totaldata);
	fprintf(log, "Latency: time per operation %lf  \n", latency);

	//printf("ReadSeqKByte: %lf ms\n", elapsed);
	elapsed=elapsed/1000;
	throughput= (double)(totaldata/elapsed);
	fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);

//--------------ReadWriteRanByte----------------------//
//printf("\nReadWriteRanByte\n");
clock_t t7, t8;
t7 = clock();
for (i = 0; i < no_of_threads; i++)
{
	pthread_create(&(tid[i]), NULL, readRanByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
	pthread_join(tid[j], NULL);
}
```

```
t8 = clock();
elapsed = timediff(t7, t8);
//printf("ReadWriteRanByte operation %lf  milliseconds\n", elapsed);
fprintf(log, "ReadWriteRanByte operation %lf  milliseconds\n", elapsed);

totaldata= (double)(no_of_threads*100000000*1)/1000000;
        latency= (double)(elapsed/totaldata);
        fprintf(log, "Latency: time per operation %lf  \n", latency);

        //printf("Throughput: operation per milliseconds %lf  \n", throughput);
        elapsed=elapsed/1000;
        throughput= (double)(totaldata/elapsed);
        fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);


//--------------ReadWriteRanKByte----------------------//
//printf("\nReadWriteRanKByte\n");
clock_t t9, t10;
t9 = clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, readRanKByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t10 = clock();
elapsed = timediff(t9, t10);
//printf("ReadWriteRanKByte operation %lf  milliseconds\n", elapsed);
fprintf(log, "ReadWriteRanKByte operation %lf  milliseconds\n", elapsed);
totaldata= (double)(no_of_threads*100000*1024)/1000000;
        latency= (double)(elapsed/totaldata);
        fprintf(log, "Latency: time per operation %lf  \n", latency);

        //printf("ReadSeqKByte: %lf ms\n", elapsed);
        elapsed=elapsed/1000;
        throughput= (double)(totaldata/elapsed);
        fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
        //printf("Throughput: operation per milliseconds %lf  \n", throughput);

//--------------ReadWriteRanMByte----------------------//
//printf("\nReadWriteRanMByte\n");
clock_t t11, t12;
t11 = clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, readRanMByte, NULL);
}
```

```
	for (j = 0; j < no_of_threads; j++)
	{
		pthread_join(tid[j], NULL);
	}
	t12 = clock();
	elapsed = timediff(t11, t12);
	//printf("ReadWriteRanMByte operation %lf  milliseconds\n", elapsed);
	fprintf(log, "ReadWriteRanMByte operation %lf  milliseconds\n", elapsed);
	totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
		latency= (double)(elapsed/totaldata);
		fprintf(log, "Latency: time per operation %lf  \n", latency);

		//printf("ReadSeqKByte: %lf ms\n", elapsed);
		elapsed=elapsed/1000;
		throughput= (double)(totaldata/elapsed);
		fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
		//printf("Throughput: operation per milliseconds %lf  \n", throughput);


	fclose(log);
   return 0;
}
```

DiskBenchmarkW.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

#define BUFFER_SIZE (1*1024*10240)
```

```
// this is buffered space we are using for reading and writing purpose form Disk and to disk

char buffer[1024*1024];


// this is to write a byte data sequencially on disk
void* writeSeqByte(void *arg)
{
	FILE *fpw;
	int i;
	fpw=fopen("DiskWrite.txt", "a");
	//char buffer[1024*1024];
	fseek(fpw,0,SEEK_SET);
	for(i=0;i<100000000;i++) // 100mb file
	{
		//fread(buffer,1,1,fp);
		fwrite(buffer,1,1,fpw);
	}
	fclose(fpw);
}

// this is to write a Kilo byte data sequencially on disk
void* writeSeqKByte(void *arg)
{
	FILE *fpw;
	int i;
	fpw=fopen("DiskWrite.txt", "a");
	//char buffer[1024*1024];
	fseek(fpw,0,SEEK_SET);
	for(i=0;i<100000;i++) // 100mb file
	{
		//fread(buffer,1,1024,fp);
		fwrite(buffer,1,1024,fpw);
	}
	fclose(fpw);
}

// this is to write a Mega byte data sequencially on disk
void* writeSeqMByte(void *arg)
{
	FILE *fpw;
	int i;
	fpw = fopen("DiskWrite.txt", "a");
	//char buffer[1024 * 1024];
	fseek(fpw, 0, SEEK_SET);
	for (i = 0; i < 100; i++) // 100mb file
	{
		//fread(buffer, 1024, 1024, fp);
		fwrite(buffer,1024,1024,fpw);
```

```
	}
	fclose(fpw);
}

// this is to write a byte data Randomly on disk
void* writeRandByte(void *arg)
{
	FILE *fpw;
	int i,r,offset;
	fpw = fopen("DiskWrite.txt", "a");


	for (i = 0; i < 100000000; i++) // 100mb file
	{
		r = rand();
		offset = r % 1000;
		fseek(fpw, offset, SEEK_SET);
		fwrite(buffer, 1, 1, fpw);
	}
	fclose(fpw);
}

// this is to write a Kilo byte data Randomly on disk
void* writeRandKByte(void *arg)
{
	FILE *fpw;
	int i,r,offset;
	fpw = fopen("DiskWrite.txt", "a");
	//char buffer[1024*1024];

	for (i = 0; i < 100000; i++) // 100mb file
	{
		r = rand();
		offset = r % 1000;
		fseek(fpw, offset, SEEK_SET);
		fwrite(buffer, 1, 1024, fpw);
	}
	fclose(fpw);
}

// this is to write a Mega byte data Randomly on disk
void* writeRandMByte(void *arg)
{
	FILE *fpw;
	int i,r,offset;
	fpw = fopen("DiskWrite.txt", "a");
	//char buffer[1024 * 1024];

	for (i = 0; i < 100; i++) // 100mb file
```

```
	{
		r = rand();
		offset = r % 1000;
		fseek(fpw, offset, SEEK_SET);
		fwrite(buffer, 1024, 1024, fpw);
	}
	fclose(fpw);
}

// Return the time difference between two clocks
long timediff(clock_t t1, clock_t t2) {
    long elapsed;
    elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC * 1000;
    return elapsed;
}


int main(int argc, char *argv[]) {

	FILE *log;
	log=fopen("Disk_log.txt", "a+");
	//fpw=fopen("Disk_Write.txt", "w+");

	int i,j,cnt=0;
	size_t size = 1;
	int number_of_operation=1;
	double elapsed;
	double latency,throughput,totaldata;
	char str[]= "Abhijeet Kedari";

	/*fseek(fp, 0, SEEK_END);
	EndPos = ftell(fp);
	fseek(fp, 0, SEEK_SET);
	StartPos = ftell(fp);
	Size_file = (EndPos - StartPos);
	printf("Size of File= %d BYTE\n\n", Size_file);*/

	int no_of_threads=atoi(argv[1]);
	pthread_t tid[no_of_threads];


	for(i=0;i<(1024*1024);i++)
	{
		buffer[i]='k';
	}


	//--------------WriteSeqByte----------------------//
	clock_t t1,t2;
```

```
	t1=clock();
	for(i=0;i<no_of_threads;i++)
	{
		pthread_create(&(tid[i]),NULL,writeSeqByte,NULL);
	}
	for(j=0;j<no_of_threads;j++)
	{
		pthread_join(tid[j],NULL);
	}
	t2=clock();
	elapsed = timediff(t1,t2);
	fprintf(log,"WriteSeqByte: %lf\n",elapsed);

	totaldata= (double)(no_of_threads*100000000*1)/1000000;
	latency= (double)(elapsed/totaldata);
	fprintf(log, "Latency: time per operation %lf  \n", latency);

	elapsed=elapsed/1000;
	throughput = (double)(totaldata/elapsed);		//As 1 thread writing 100 mb, 4 threads are writing 400 data...
	fprintf(log, "Throughput: operations(mb) per seconds %lf  \n", throughput);


	//--------------WriteSeqKByte-----------------------//
	clock_t t3,t4;
	t3=clock();
	for (i = 0; i < no_of_threads; i++)
	{
		pthread_create(&(tid[i]), NULL, writeSeqKByte, NULL);
	}
	for (j = 0; j < no_of_threads; j++)
	{
		pthread_join(tid[j], NULL);
	}
	t4=clock();
	elapsed = timediff(t3,t4);
	fprintf(log,"WriteSeqKByte: %lf\n",elapsed);

	totaldata= (double)(no_of_threads*100000*1024)/1000000;
	latency=(double)(elapsed/totaldata);
	fprintf(log, "Latency: time per operation %lf  \n", latency);

	elapsed =elapsed /1000;
	throughput = (double)(totaldata)/elapsed;
	//printf("Throughput: operation per milliseconds %lf  \n", throughput);
	//printf("\n totaldata= %lf",totaldata);
	fprintf(log, "Throughput: operations[mb] per seconds %lf  \n", throughput);
```

```
	//--------------WriteSeqMByte----------------------//
clock_t t5,t6;
t5=clock();
for (i = 0; i < no_of_threads; i++)
{
	pthread_create(&(tid[i]), NULL, writeSeqMByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
	pthread_join(tid[j], NULL);
}
t6=clock();
elapsed = timediff(t5,t6);
fprintf(log,"WriteSeqMByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
latency=(double)(elapsed/totaldata);
fprintf(log, "Latency: time per operation %lf  \n", latency);

elapsed =elapsed /1000;
throughput = (double)(totaldata/elapsed);
//printf("Throughput: operation(mb) per seconds %lf  \n", throughput);
//printf("\n totaldata= %lf",totaldata);
fprintf(log, "Throughput: operations(mb) per seconds %lf  \n", throughput);

//--------------WriteRandByte----------------------//
clock_t t7,t8;
t7=clock();
for(i=0;i<no_of_threads;i++)
{
	pthread_create(&(tid[i]),NULL,writeRandByte,NULL);
}
for(j=0;j<no_of_threads;j++)
{
	pthread_join(tid[j],NULL);
}
t8 = clock();
elapsed = timediff(t7, t8);
fprintf(log,"\tWriteRandByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100000000*1)/1000000;
latency=(double)(elapsed/totaldata);
fprintf(log, "\tLatency: time per operation %lf  \n", latency);

elapsed =elapsed /1000;
throughput = (double)(totaldata/elapsed);
fprintf(log, "\tThroughput: operations per milliseconds %lf  \n", throughput);
```

```
//--------------WriteRandKByte-----------------------//
clock_t t9,t10;
t9=clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, writeRandKByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t10 = clock();
elapsed = timediff(t9, t10);
fprintf(log,"\tWriteRandKByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100000*1024)/1000000;
latency=(double)(elapsed/totaldata);
fprintf(log, "\tLatency: time per operation %lf  \n", latency);
//printf("WriteRandKByte: %lf ms\n", elapsed);

elapsed =elapsed /1000;
throughput =(double)(totaldata/elapsed);
//printf("Throughput: operation per milliseconds %lf  \n", throughput);
fprintf(log, "\tThroughput: operations per milliseconds %lf  \n", throughput);


//--------------WriteRandMByte-----------------------//
clock_t t11,t12;
t11=clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, writeRandMByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t12 = clock();
elapsed = timediff(t11, t12);
fprintf(log,"\tWriteRandMByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
latency=(double)(elapsed/totaldata);
fprintf(log, "\tLatency: time per operation %lf  \n", latency);

//printf("WriteRandMByte: %lf ms\n", elapsed);
elapsed =elapsed /1000;
throughput = (double)(totaldata/elapsed);
//printf("Throughput: operation per milliseconds %lf  \n", throughput);
```

```
        fprintf(log, "\tThroughput: operations per milliseconds %lf  \n", throughput);

        fclose(log);
    return 0;


}
```

DiskBenchmarkR.c

```
/*

============================================================================
===
 Name        : Programming.c
 Author      : Abhijeet
 Version     :
 Copyright   : Your copyright notice
 Description : Hello World in C, Ansi-style

============================================================================
===
 */

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

#define BUFFER_SIZE (1*1024*10240)
// this is buffered space we are using for reading and writing purpose form Disk and to disk
char buffer[1*1024*1024];

// this is to Read a byte data sequencially from disk
void* readSeqByte(void *arg)
{
        FILE *fp;
        int i;
        fp=fopen("DiskWrite.txt", "r");
        char buffer[1024*1024];
        fseek(fp,0,SEEK_SET);
        for(i=0;i<100000000;i++) // 100mb file
        {
                fread(buffer,1,1,fp);
        }
```

```
	fclose(fp);
}

// this is to Read a Kilo byte data sequencially from disk
void* readSeqKByte(void *arg)
{
	FILE *fp;
	int i;
	fp=fopen("DiskWrite.txt", "r");
	char buffer[1024*1024];
	fseek(fp,0,SEEK_SET);
	for(i=0;i<100000;i++) // 100mb file
	{
		fread(buffer,1,1024,fp);
	}
	fclose(fp);
}

// this is to Read a mega byte data sequencially from disk
void* readSeqMByte(void *arg)
{
	FILE *fp;
	int i;
	fp = fopen("DiskWrite.txt", "r");
	char buffer[1024 * 1024];
	fseek(fp, 0, SEEK_SET);
	for (i = 0; i < 100; i++) // 100mb file
	{
		fread(buffer, 1024, 1024, fp);
	}
	fclose(fp);
}

// this is to Read a byte data Randomly from disk
void* readRandByte(void *arg)
{
	FILE *fp;
	int i,r,offset;
	fp = fopen("DiskWrite.txt", "r");
	char buffer[1024 * 1024];

	int size_of_file=*(int *)arg;

	for (i = 0; i < 10000000; i++) // 10mb file
	{
		r=rand();
		offset = r % size_of_file;
		fseek(fp, offset, SEEK_SET);
		fread(buffer, 1, 1, fp);
```

```
	}
	fclose(fp);
}

// this is to Read a kilo byte data Randomly from disk
void* readRandKByte(void *arg)
{
	FILE *fp;
	int i,r,offset;
	fp=fopen("DiskWrite.txt", "r");
	char buffer[1024*1024];

	int size_of_file=*(int *)arg;
	size_of_file= size_of_file - 1024;

	//fseek(fp,0,SEEK_SET);
	for(i=0;i<10000;i++) // 10mb file
	{
		r = rand();
		offset = r % size_of_file;
		fseek(fp, offset, SEEK_SET);
		fread(buffer,1,1024,fp);
	}
	fclose(fp);
}

// this is to Read a Mega byte data Randomly from disk
void* readRandMByte(void *arg)
{
	FILE *fp;
	int i,r,offset;
	fp = fopen("DiskWrite.txt", "r");
	char buffer[1024 * 1024];

	int size_of_file=*(int *)arg;
	size_of_file= size_of_file - (1024*1024);

	for (i = 0; i < 10; i++) // 1mb file
	{
		r = rand();
		offset = r % size_of_file;
		fseek(fp, offset, SEEK_SET);
		fread(buffer, 1024, 1024, fp);
	}
	fclose(fp);
}

// Return the time difference between two clocks
double timediff(clock_t t1, clock_t t2) {
```

```
        double elapsed;
    elapsed = ((double)t2 - t1) / CLOCKS_PER_SEC * 1000;
    return elapsed;
}


int main(int argc, char *argv[]) {

        FILE *fp,*log;
        fp=fopen("DiskWrite.txt", "r");
        log=fopen("Disk_log.txt", "a+");

        int i,j,cnt=0,EndPos,StartPos,Size_file;
        size_t size = 1;
        double latency,throughput,totaldata,elapsed;
        int number_of_operation=1;

        char str[]= "Abhijeet Kedari";

        fseek(fp, 0, SEEK_END);
        EndPos=ftell(fp);
        fseek(fp, 0, SEEK_SET);
        StartPos=ftell(fp);
        Size_file = (EndPos - StartPos);

        int no_of_threads=atoi(argv[1]);
        pthread_t tid[no_of_threads];


        //--------------ReadSeqByte-----------------------//
        clock_t t1,t2;
        t1=clock();
        for(i=0;i<no_of_threads;i++)
        {
                pthread_create(&(tid[i]),NULL,readSeqByte,NULL);
        }
        for(j=0;j<no_of_threads;j++)
        {
                pthread_join(tid[j],NULL);
        }
        t2=clock();
        elapsed = timediff(t1,t2);
        fprintf(log,"ReadSeqByte: %lf\n",elapsed);

        //printf("ReadSeqByte: %lf ms\n", elapsed);
        totaldata= (double)(no_of_threads*100000000*1)/1000000;
        latency= (double)(elapsed/totaldata);
        fprintf(log, "Latency: time per operation %lf  \n", latency);
```

```
//printf("Throughput: operation per milliseconds %lf  \n", throughput);
elapsed=elapsed/1000;
throughput= (double)(totaldata/elapsed);
fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);

//--------------ReadSeqKByte----------------------//
clock_t t3,t4;
t3=clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, readSeqKByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t4=clock();
elapsed = timediff(t3,t4);
fprintf(log,"ReadSeqKByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100000*1024)/1000000;
latency= (double)(elapsed/totaldata);
fprintf(log, "Latency: time per operation %lf  \n", latency);

//printf("ReadSeqKByte: %lf ms\n", elapsed);
elapsed=elapsed/1000;
throughput= (double)(totaldata/elapsed);
fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
//printf("Throughput: operation per milliseconds %lf  \n", throughput);


//--------------ReadSeqMByte----------------------//
clock_t t5,t6;
t5=clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, readSeqMByte, NULL);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t6=clock();
elapsed = timediff(t5,t6);
fprintf(log,"ReadSeqMByte: %lf\n",elapsed);

totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
latency= (double)(elapsed/totaldata);
fprintf(log, "Latency: time per operation %lf  \n", latency);
```

```
//printf("ReadSeqKByte: %lf ms\n", elapsed);
elapsed=elapsed/1000;
throughput= (double)(totaldata/elapsed);
fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
//printf("Throughput: operation per milliseconds %lf  \n", throughput);



//--------------ReadRandByte-----------------------//
clock_t t7,t8;
t7=clock();
for(i=0;i<no_of_threads;i++)
{
        pthread_create(&(tid[i]),NULL,readRandByte,&Size_file);
}
for(j=0;j<no_of_threads;j++)
{
        pthread_join(tid[j],NULL);
}
t8 = clock();
elapsed = timediff(t7, t8);
//printf("ReadRandByte: %lf ms\n", elapsed);
fprintf(log,"\tReadRandByte: %lf\n",elapsed);
//printf("ReadSeqByte: %lf ms\n", elapsed);

totaldata= (double)(no_of_threads*100000000*1)/1000000;
latency= (double)(elapsed/totaldata);
fprintf(log, "Latency: time per operation %lf  \n", latency);

//printf("Throughput: operation per milliseconds %lf  \n", throughput);
elapsed=elapsed/1000;
throughput= (double)(totaldata/elapsed);
fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);


//--------------ReadRandKByte-----------------------//
clock_t t9,t10;
t9=clock();
for (i = 0; i < no_of_threads; i++)
{
        pthread_create(&(tid[i]), NULL, readRandKByte, &Size_file);
}
for (j = 0; j < no_of_threads; j++)
{
        pthread_join(tid[j], NULL);
}
t10 = clock();
elapsed = timediff(t9, t10);
```

```
        fprintf(log,"\tReadRanKByte: %lf\n",elapsed);

        totaldata= (double)(no_of_threads*100000*1024)/1000000;
        latency= (double)(elapsed/totaldata);
        fprintf(log, "Latency: time per operation %lf  \n", latency);

        //printf("ReadSeqKByte: %lf ms\n", elapsed);
        elapsed=elapsed/1000;
        throughput= (double)(totaldata/elapsed);
        fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
        //printf("Throughput: operation per milliseconds %lf  \n", throughput);


        //--------------ReadRandMByte----------------------//
        clock_t t11,t12;
        t11=clock();
        for (i = 0; i < no_of_threads; i++)
        {
                pthread_create(&(tid[i]), NULL, readRandMByte, &Size_file);
        }
        for (j = 0; j < no_of_threads; j++)
        {
                pthread_join(tid[j], NULL);
        }
        t12 = clock();
        elapsed = timediff(t11, t12);
        fprintf(log,"\tReadRandByte: %lf\n",elapsed);

        totaldata= (double)(no_of_threads*100*1024*1024)/1000000;
        latency= (double)(elapsed/totaldata);
        fprintf(log, "Latency: time per operation %lf  \n", latency);

        //printf("ReadSeqKByte: %lf ms\n", elapsed);
        elapsed=elapsed/1000;
        throughput= (double)(totaldata/elapsed);
        fprintf(log,"Throughput: operations(mb) per seconds %lf  \n", throughput);
        //printf("Throughput: operation per milliseconds %lf  \n", throughput);

    fclose(log);
    return 0;


}
```