Report

Timing results :

N	B	P	C	Time
100	4	1	1	0.000854
100	4	1	2	0.001690
100	4	1	3	0.003931
100	4	2	1	0.003207
100	4	3	1	0.003833
100	8	1	1	0.002566
100	8	1	2	0.003300
100	8	1	3	0.003935
100	8	2	1	0.003125
100	8	3	1	0.003697
398	8	1	1	0.003775
398	8	1	2	0.004344
398	8	1	3	0.004653
398	8	2	1	0.004363
398	8	3	1	0.005093

N	B	P	C	Time
100	4	1	1	0.001221
100	4	1	2	0.001685
100	4	1	3	0.000467
100	4	2	1	0.000413
100	4	3	1	0.000474
100	8	1	1	0.000335
100	8	1	2	0.000432
100	8	1	3	0.000472
100	8	2	1	0.000449
100	8	3	1	0.000652
398	8	1	1	0.003449
398	8	1	2	0.001945
398	8	1	3	0.000581
398	8	2	1	0.000468
398	8	3	1	0.000684

Given (N,B,P,C) = (398, 8, 1, 3),
Average system execution time = 0.005093 s
Standard deviation of system execution time = 0.000684 s

Appendix:

producer.c:

```
#include <mqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <signal.h>
#include <unistd.h>
#include <sys/time.h>


#define _XOPEN_SOURCE 600

int i;
//----------------- PRODUCER -----------------
int main(int argc, char *argv[])
{
	mqd_t qdes;
 //The real mailbox. The messages are sent to this mailbox and also received from this mailbox
	char  qname[] = "/mailboxLab5";
	mqd_t qdes2;
//This mailbox is to control how many items were produced and consumed.
	char  qname2[] = "/mb_control";
	mode_t mode = S_IRUSR | S_IWUSR;
	struct mq_attr attr;

	attr.mq_maxmsg  = atoi(argv[2]); //Receiving "b" from producer
	attr.mq_msgsize = sizeof(int); // The size of each message
	attr.mq_flags   = 0;    // a blocking queue

//Opening mailboxLab5
	qdes  = mq_open(qname, O_RDWR | O_CREAT, mode, &attr);
	if (qdes == -1 ) {
		perror("mq_open() failed");
		exit(1);
	}

//Opening the correct mailbox
	qdes2  = mq_open(qname2, O_RDONLY, mode, &attr);
	if (qdes == -1 ) {
		perror("mq_open() failed");
		exit(1);
	}

	int rcv; //Just to 'receive' the messages of the mb_control
```

```
 //Sending message to mailboxLab5
// argv[4] = pid  argv[1]=N  argv[3]=P
      for(i=atoi(argv[4]); i<atoi(argv[1]); i=i+atoi(argv[3])){
         //Receiving message from mb_control ---- THIS PART IS LIKE A SEMAPHORE
          if (mq_receive(qdes2, (char *)&rcv , sizeof(int), 0) == -1)
              perror("mq_receive() failed ");

          //adding iten to the mailboxLab5
          if (mq_send(qdes, (char *)&i , sizeof(int), 0) == -1)
               perror("mq_send() failed ");
     }

    //Closing the mailboxes and checking for errors in the process
      if (mq_close(qdes) == -1) {
            perror("mq_close() failed");
            exit(2);
      }

      if (mq_close(qdes2) == -1) {
            perror("mq_close() failed");
            exit(2);
      }

     return 0;
}
```

consumer.c:

```
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <mqueue.h>
#include <sys/stat.h>
#include <signal.h>
#include <unistd.h>
#include <sys/time.h>


#define _XOPEN_SOURCE 600

int i;
//------------------- CONSUMER --------------------
int main(int argc, char *argv[])
{
	mqd_t qdes;
	char  qname[] = "/mailboxLab5"; //mailbox with the real messages
	mqd_t qdes2;
//mailbox to control the itens produced and consumed
	char  qname2[] = "/mb_control";
	mode_t mode = S_IRUSR | S_IWUSR;
	struct mq_attr attr;

	attr.mq_maxmsg  = atoi(argv[2]); //Receiving "b" from producer
	attr.mq_msgsize = sizeof(int); // The size of each message
	attr.mq_flags   = 0;    // a blocking queue

	qdes  = mq_open(qname, O_RDONLY, mode, &attr); //Opening mailboxLab5
	if (qdes == -1 ) {
		perror("mq_open()");
		printf("mq error inside CONSUMER %d\n", getpid());
		exit(1);
	}

	qdes2  = mq_open(qname2, O_RDWR | O_CREAT, mode, &attr); //Opening
mb_control
	if (qdes2 == -1 ) {
		perror("mq_open()");
		printf("mq error inside CONSUMER %d\n", getpid());
		exit(1);
	}
```

```
	int send = 0;

//Same loop as the producer. Just the last argument is different (here it is
argv[3]=C). Each consumer consumes their respective amounts.
	for(i=atoi(argv[4]) ; i<atoi(argv[1]) ; i=i+atoi(argv[3])) {
	   int int_num;
	   if (mq_receive(qdes, (char *)&int_num, sizeof(int), 0) == -1) {
	      perror("mailbox failed");
	   }
	   else {
	      double sqroot = sqrt(int_num);
//Checking if the root of the received message is an integer
	      if(sqroot/1.00 == (int)sqroot)
	         printf("%d %d %d\n",atoi(argv[4]),int_num,(int)sqroot);
	      if(mq_send(qdes2,(char*)&send, sizeof(int), 0) == -1)
	         perror("mailbox failed");
	   }
	}

	//Closing the mailboxes and checking for errors in the process
	if (mq_close(qdes) == -1) {
	      perror("mq_close() failed");
	      exit(2);
	}

	if (mq_close(qdes2) == -1) {
	      perror("mq_close() failed");
	      exit(2);
	}

	return 0;
}
```

produce.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <mqueue.h>
#include <sys/stat.h>
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <semaphore.h>


double t1, t2; //initializing the timing variables
struct timeval tv;
sem_t items_counter;
sem_t binary_sem;
pid_t ch_pid[50];

//------ CONSUMER SPAWN ------
int spawn (char* program, char* n, char* b, char* c, int id, char* p)
{
 pid_t child_pid;
 char char_id[6];
 snprintf(char_id, 6," %d", id);
 child_pid = fork ();
 ch_pid[id+atoi(p)] = child_pid;

 char* arg_list[] = {
    "consumer", //argv[0], the name of the program
    n,    //The number of items to be produced by the producer
    b,    //The size of the queue mailbox
    c,   //Number of consumers
    char_id,   //pid
    NULL //the argument list MUST end with NULL
 };

 if (child_pid != 0) {
  /* This is the parent process.  */
   return child_pid;
 }
 else {
  /* Now execute PROGRAM, searching for it in the path.  */
  execvp (program, arg_list);
  /* The execvp function returns only if an error occurs.  */
```

```
   fprintf (stderr, "an error occurred in execvp in consumer\n");
   abort ();
  }
}


//------ PRODUCER SPAWN -----
int spawn2 (char* program, char* n, char* b, char* p, int id)
{
  pid_t child_pid;
  char char_id[6];
  snprintf(char_id, 6," %d", id); //transforning integer into char
  child_pid = fork ();
  ch_pid[id] = child_pid;

  //arg_list contains a list of arguments that will be passed to  the consumer
  char* arg_list[] = {
     "producer", //argv[0], the name of the program
     n, //The size of the queue mailbox
     b, //The number of items to be produced by the producer
     p,
     char_id,
     NULL //the argument list MUST end with NULL
  };

  if (child_pid != 0) {
    /* This is the parent process.  */
    return child_pid;
 }
  else {
    /* Now execute PROGRAM, searching for it in the path.  */
    execvp (program, arg_list);
    /* The execvp function returns only if an error occurs.  */
    fprintf (stderr, "an error occurred in execvp in producer\n");
    abort ();
  }
}



int main(int argc, char *argv[])
{
       int child_status;
       pid_t pid;
       mqd_t qdes;
       mqd_t qdes2;
```

```
    char  qname2[] = "/mb_control";
    char  qname[] = "/mailboxLab5"; //queue name must start with '/'
    mode_t mode = S_IRUSR | S_IWUSR;
    struct mq_attr attr;
    int j;
    char  qname2[] = "/mb_control";
    char  qname[] = "/mailboxLab5"; //queue name must start with '/'
    mode_t mode = S_IRUSR | S_IWUSR;
    struct mq_attr attr;
    int j;
    int a[50];

    if ( argc !=5 ) {
        printf("You have to enter: ./produce <N> <B>\n");
        printf("<N> = number of integers the producer should produce\n");
        printf("<B> = number of integers the message queue can hold\n");
        printf("<P> = number of producers\n");
        printf("<C> = number of consumers\n");
        exit(1);
    }

    attr.mq_maxmsg  = atoi(argv[2]); //Receiving "b" from producer
    attr.mq_msgsize = sizeof(int); // The size of each message
    attr.mq_flags   = 0;    // a blocking queue

    qdes  = mq_open(qname, O_RDWR | O_CREAT, mode, &attr); //Opening the
mailboxLab5
    if (qdes == -1 ) {
        perror("mq_open()");
        printf("mq error inside MAIN  %d\n", getpid());
        exit(1);
    }

//Opening the control mailbox - mb_control
    qdes2  = mq_open(qname2, O_RDWR | O_CREAT, mode, &attr);
    if (qdes2 == -1 ) {
        perror("mq_open()");
        printf("mq error inside MAIN  %d\n", getpid());
        exit(1);
    }

      int msg_null = 0;
    for(j=0; j<atoi(argv[2]); j++){
      if(mq_send(qdes2, (char*)&msg_null, sizeof(int), 0) == -1){
        sleep(1);
        perror("mq_send() failed");
```

```
      }
     }

     gettimeofday(&tv, NULL);
     t1 = tv.tv_sec + tv.tv_usec/1000000.0; //Time before creating first process

     for(j=0; j<atoi(argv[3]);j++) { //loops till the number of producers
      a[j]=j;
      if(spawn2("./producer.out", argv[1], argv[2],argv[3],a[j]) != 0) {
       continue; // This is to continue doing the fork when it is the parent process
      }
     }

     for(j=0; j<atoi(argv[4]);j++) { //loops till the number of consumers
      a[j+atoi(argv[3])]=j;
      if(spawn("./consumer.out", argv[1], argv[2],argv[4],a[j+atoi(argv[3])],argv[3])
!= 0) {
       continue; // This is to continue doing the fork when it is the parent process
      }
     }

    //Calling waitpid. So the main function can wait all the processes finish, and
then it can finish
    for(j=0; j<(atoi(argv[3]) + atoi(argv[4])); j++){
      waitpid(ch_pid[j],&child_status,0);
      if(WIFEXITED(child_status)){
      } else
       printf("%d was  NOT waited  %d\n",j,ch_pid[j]);
    }

     if (mq_close(qdes) == -1) {
         perror("mq_close() failed inside MAIN");
         exit(2);
     }

     if (mq_close(qdes2) == -1) {
         perror("mq_close() failed inside MAIN");
         exit(2);
     }
     if (mq_unlink(qname) != 0) {
      perror("mq_unlink() failed inside MAIN");
         exit(3);
     }
     if (mq_unlink(qname2) != 0) {
      perror("mq_unlink() failed inside MAIN");
         exit(3);
```

```
    }

    gettimeofday(&tv, NULL);
    t2 = tv.tv_sec + tv.tv_usec/1000000.0; //Time after last integer is consumed
//Calculating the system execution time
    printf("System execution time: %f seconds\n", t2-t1);
    return 0;
}
```

By comparing the timing results for the multi-thread with shared memory and the multi-process with message queue approach, we notice that the multi-thread approach has a smaller execution time. This could be because forking a process usually takes some time where as in threads since all threads share the same memory, it executes faster.

The advantages and disadvantages of multi-thread approach:
Advantages:

1. It takes lesser time to create a new thread.
2. It takes lesser time to switch between two threads in a process than to switch between processes because the threads share the same memory.
3. Data sharing with other threads in a process: for tasks that require sharing large amounts of data, the fact that threads all share a process's memory pool is very beneficial. Not having separate copies means that different threads can read and modify a shared pool of memory easily. While data sharing is possible with separate processes through shared memory and inter-process communication, this sharing is of an arms-length nature and is not inherently built into the process model.

Disadvantages:

1. Synchronization overhead of shared data: shared data that is modified requires special handling in the form of locks, mutexes and other primitives to ensure that data is not being read while written, nor written by multiple threads at the same time.
2. All threads in a process share the same memory space. If something goes wrong in one thread and causes data corruption or an access violation, then this affects and corrupts all the threads in that process.

The advantages and disadvantages of multi-process with message queue approach:
Advantages:

1. Shared data that is modified does not require special handling in the form of locks, mutexes and other primitives.
2. Since the processes do not share memory, if something goes wrong in one process and causes data corruption or an access violation, then this does not affect and corrupts all the processes. Hence processes are more secure.

Disadvantages:

1. It takes more time to fork a new process than to create a new thread.
2. Since we have to send and receive messages using a queue, this causes a significant delay and there is a chance that the data may be transmitted with errors.