Syed Waqar Uddin

CEG 4350/ 6350 Project

April 26th, 2019

REPORT

1. IPC Method: Message Passing   Language: C programming with Ubuntu's default text editor   OS: Ubuntu
   i. Description of Design:

I used Ubuntu's default text editor using C programming language to design the code. There are 2 files. One is producerProcess.C and the other is consumerProcess.C. In order to run the two files on the terminal type the command "gcc -o producer producerProcess.C". Then "./producer". This will create the file producerData.txt containing the 100 random integer. After this run them command "gcc -o consumer consumerProcess.c. Then "./consumer". This will create a file consumerProcess.txt containing the 100 random integers read by the consumer.

For the producer process the system call used is 'key = ftok("progfile", 65);'this will generate a unique key. After this I used 'msgid = msgget(key, 0666 | IPC_CREAT); 'which will give the read and write access to the to the process, finally a message queue will be created, containing that unique key generated in the previous step. Then a ProdcuerData.txt file will be opened 'fp = fopen("ProducerData.txt", "w+");' where the 10 random integers generated 'message.mesg_text= rand() % 100 + 1;' by the producer will be stored.

For consumer process also I used the 'key = ftok("progfile", 65);' to generate the unique key, then ' msgid = msgget(key, 0666 | IPC_CREAT);' to create the message queue. The 'ConsumerData.txt' file is created to store the integers read by the consumer process to receive the message 'msgrcv(msgid, &message, sizeof(message), 1, 0); '

..

   ii. Source Code:

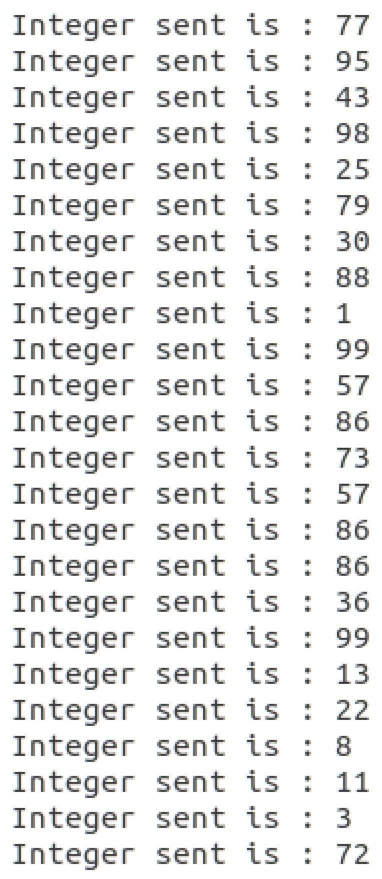
(producerProcess.c)

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <time.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    int mesg_text;


} message;

int main()
{
    key_t key;
    int msgid;
int c;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;
FILE *fp;
fp = fopen("ProducerData.txt", "w+");
   // Print 100 random integers on the console
srand(time(0));
for (c = 1; c <= 100; c++) {

    message.mesg_text= rand() % 100 + 1;
   msgsnd(msgid, &message, sizeof(message), 0);
fprintf(fp," Integer sent is : %d\n", message.mesg_text);

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

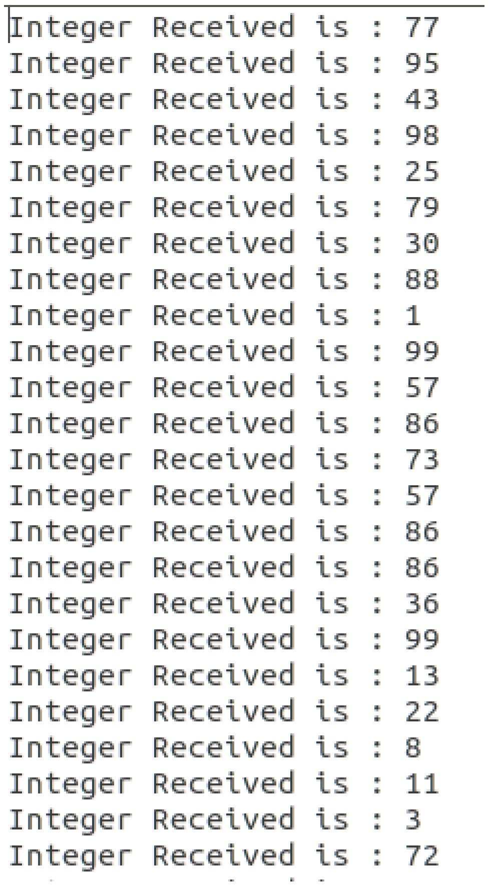
(consumerProcess.c)

```c
// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    int mesg_text;
} message;

int main()
{
    key_t key;
    int msgid;
    int c;

FILE *fp;
fp = fopen("ConsumerData.txt", "w+");

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

for (c =1; c<= 100; c++){

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    fprintf(fp,"Integer Received is : %d\n",
                    message.mesg_text);
  }
fclose(fp);
    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

iii. Result:



```
Integer sent is : 77
Integer sent is : 95
Integer sent is : 43
Integer sent is : 98
Integer sent is : 25
Integer sent is : 79
Integer sent is : 30
Integer sent is : 88
Integer sent is : 1
Integer sent is : 99
Integer sent is : 57
Integer sent is : 86
Integer sent is : 73
Integer sent is : 57
Integer sent is : 86
Integer sent is : 86
Integer sent is : 36
Integer sent is : 99
Integer sent is : 13
Integer sent is : 22
Integer sent is : 8
Integer sent is : 11
Integer sent is : 3
Integer sent is : 72
```


```
Integer Received is : 77
Integer Received is : 95
Integer Received is : 43
Integer Received is : 98
Integer Received is : 25
Integer Received is : 79
Integer Received is : 30
Integer Received is : 88
Integer Received is : 1
Integer Received is : 99
Integer Received is : 57
Integer Received is : 86
Integer Received is : 73
Integer Received is : 57
Integer Received is : 86
Integer Received is : 86
Integer Received is : 36
Integer Received is : 99
Integer Received is : 13
Integer Received is : 22
Integer Received is : 8
Integer Received is : 11
Integer Received is : 3
Integer Received is : 72
```

iv. Discussion:

For this project I was using Ubuntu for the first time and I got to learn a great deal from it. I found that the code was far more convenient to build on Ubuntu compared to the Windows. I technical thing I learned is the use of IPC_CREAT 0666. This system call tells the memory segment to create a shared memory segment. IPC_CREAT | 006 for the producer means creating and granting read and write access while for the consumer this means granting read and write access.

2. IPC Method : Pipe Language : Bash scripting OS: Ubuntu

i. Description of the design:

First 'if [! -f $pipe]' this makes sure that there is no existing pipe.In the producer process I used 'mkfifo' to create the pipe. array = ( ) to initialize the array. The 100 random integers generated are stored in the array and then they are copied to the pipe. In consumer.sh the consumer process checks whether there is already existing pipe. If there is no pipe then the process waits for 5 second for the pipe to be created. If there is a pipe then consumer process reads the pipe line by line and prints to the console.

ii. Source Code:

(Producer.sh):

```
#!/bin/sh

pipe=/home/testpipe

array=()

trap "rm -f $pipe" EXIT #precaution to remove the pipe when exited or signal interrupted
```

```
if [ ! -f $pipe ]; then
   mkfifo $pipe
   echo "named pipe created..."
   echo "Producer integers.."
fi

for i in {1..100}
do
   array+=($RANDOM) #push random numbers to array
done
echo ${array[*]}
echo ${array[*]} > $pipe #writing the array to pipe
echo "copied 100 random numbers to pipe"
exit 0
```

(Consumer.sh):

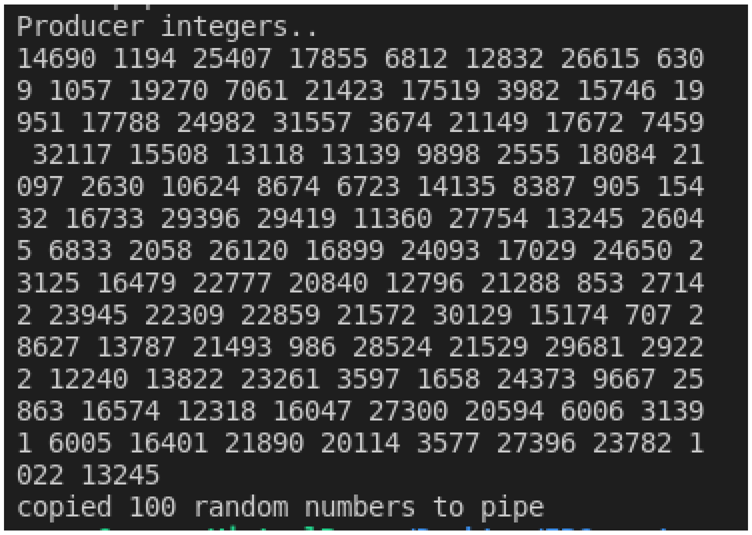
```
pipe=/home/testpipe
while true;
do
   #if file exists break the loop and print
   if [[ -p $pipe ]]; then
        break
   fi
   echo "waiting for the pipe to start, will recheck in 5 seconds"
   sleep 5
done
#print line by line #also can do cat < $pipe to print the whole file
while read line <$pipe
do
   if [[ "$line" == 'quit' ]]; then
      break
   fi
   echo "Consumer reading integers"
```

echo $line

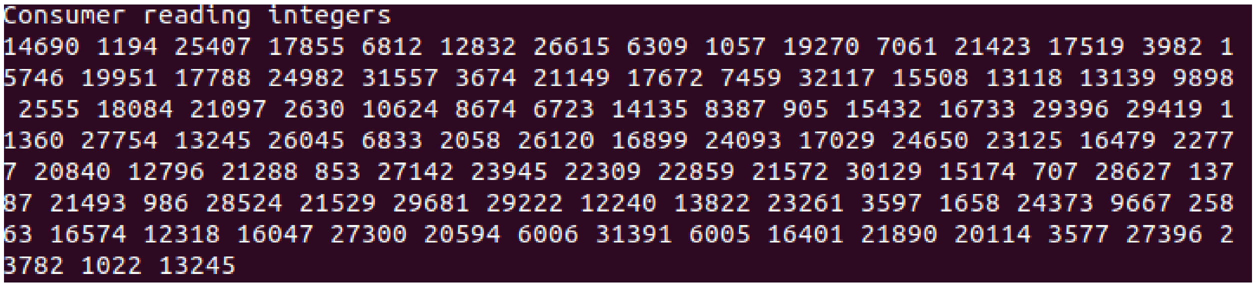
done

exit 0

iii. Result:



```
Producer integers..
14690 1194 25407 17855 6812 12832 26615 630
9 1057 19270 7061 21423 17519 3982 15746 19
951 17788 24982 31557 3674 21149 17672 7459
 32117 15508 13118 13139 9898 2555 18084 21
097 2630 10624 8674 6723 14135 8387 905 154
32 16733 29396 29419 11360 27754 13245 2604
5 6833 2058 26120 16899 24093 17029 24650 2
3125 16479 22777 20840 12796 21288 853 2714
2 23945 22309 22859 21572 30129 15174 707 2
8627 13787 21493 986 28524 21529 29681 2922
2 12240 13822 23261 3597 1658 24373 9667 25
863 16574 12318 16047 27300 20594 6006 3139
1 6005 16401 21890 20114 3577 27396 23782 1
022 13245
copied 100 random numbers to pipe
```



```
Consumer reading integers
14690 1194 25407 17855 6812 12832 26615 6309 1057 19270 7061 21423 17519 3982 1
5746 19951 17788 24982 31557 3674 21149 17672 7459 32117 15508 13118 13139 9898
 2555 18084 21097 2630 10624 8674 6723 14135 8387 905 15432 16733 29396 29419 1
1360 27754 13245 26045 6833 2058 26120 16899 24093 17029 24650 23125 16479 2277
7 20840 12796 21288 853 27142 23945 22309 22859 21572 30129 15174 707 28627 137
87 21493 986 28524 21529 29681 29222 12240 13822 23261 3597 1658 24373 9667 258
63 16574 12318 16047 27300 20594 6006 31391 6005 16401 21890 20114 3577 27396 2
3782 1022 13245
```

iv. Discussion:

I did this in Bash scripting. This language was very new for me, but I learned a lot of the commands by doing this project. The commands used in bash scripting are very different then what we are used to but it definitely a valuable skill to have. I realized that pipe is used as a kind of a file over here. Where the producer writes to the file or pipe and then the consumer reads it from the other end of the pipe or a file.

3. IPC Method : Semaphores   Language : JAVA   OS: Windows.

i. Description of Design:

To run this code first of all this command should be run “Javac *.java”. After this the Main.java could be easily complied. The code has been designed in a way so that only one process can access the resource at a time. After a process is done using the shared resource the lock should be released so that other process can use the resource.

First of all producer process is created where the random integers generated are stored in the list using Producer.list.add (int)(Math.random())*100;. The system call used is the thread to create the Handler thread class. HandlerThread consumer1 = new HandlerThread(sem, " CONSUMER1 "); to create consumer 1 process.

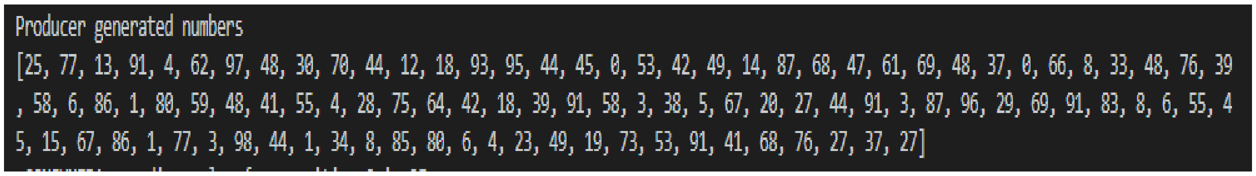
HandlerThread consumer2 = new HandlerThread(sem, " CONSUMER2 "); to create consumer 2 process.
. When consumer1 is reading the random numbers from the list consumer2 cannot read the integers from the list. Once consumer1 is done reading then only consumer2 can read the integers.

ii. Source Code:

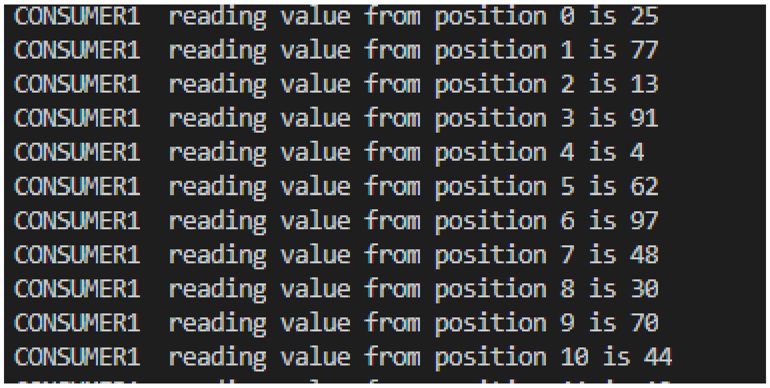
```
import java.util.LinkedList;

import java.util.Random;

import java.util.Scanner;

import java.util.concurrent.*;

import java.lang.Math;


class Producer {

   static int count = 0;

   static LinkedList list = new LinkedList();

   static Integer loop = 100;


   public static void generate() {

       try {

           for(int i=0; i< Producer.loop; i++) {

              Producer.list.add((int)(Math.random() *100));

        }

        System.out.println(Producer.list);



       } catch (Exception exc) {

           System.out.println(exc);

       }

   }

}


class HandlerThread extends Thread {

   Semaphore sem;

   Random random = new Random();
```

```
    public HandlerThread(Semaphore sem, String threadName) {
        super(threadName);
        this.sem = sem;
    }

    @Override
    public void run() {
        try {
            sem.acquire(); // semaphore activates and thread waits till it gets released
            for (int i = 0; i < Producer.loop; i++) {
                System.out.println(this.getName() + " reading value from position " + i + " is " + Producer.list.get(i));
            }

        } catch (InterruptedException exc) {
            System.out.println(exc);
        }
        System.out.println(this.getName() + " releases the permit.");
        sem.release();
    }

}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Semaphore sem = new Semaphore(1); // only one consumer allowed at a time
        Producer.generate();
        HandlerThread consumer1 = new HandlerThread(sem, " CONSUMER1 ");
        HandlerThread consumer2 = new HandlerThread(sem, " CONSUMER2 ");
        consumer1.start();
        consumer2.start();
        consumer1.join();
        consumer2.join();
    }
}
```

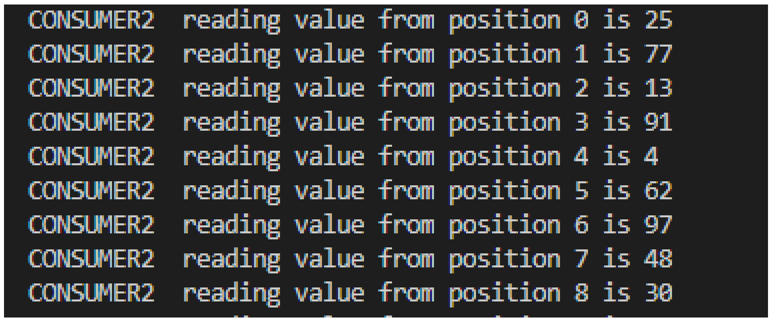
iii. Results:



```
Producer generated numbers
[25, 77, 13, 91, 4, 62, 97, 48, 30, 70, 44, 12, 18, 93, 95, 44, 45, 0, 53, 42, 49, 14, 87, 68, 47, 61, 69, 48, 37, 0, 66, 8, 33, 48, 76, 39
, 58, 6, 86, 1, 80, 59, 48, 41, 55, 4, 28, 75, 64, 42, 18, 39, 91, 58, 3, 38, 5, 67, 20, 27, 44, 91, 3, 87, 96, 29, 69, 91, 83, 8, 6, 55, 4
5, 15, 67, 86, 1, 77, 3, 98, 44, 1, 34, 8, 85, 80, 6, 4, 23, 49, 19, 73, 53, 91, 41, 68, 76, 27, 37, 27]
```



```
CONSUMER1  reading value from position 0 is 25
CONSUMER1  reading value from position 1 is 77
CONSUMER1  reading value from position 2 is 13
CONSUMER1  reading value from position 3 is 91
CONSUMER1  reading value from position 4 is 4
CONSUMER1  reading value from position 5 is 62
CONSUMER1  reading value from position 6 is 97
CONSUMER1  reading value from position 7 is 48
CONSUMER1  reading value from position 8 is 30
CONSUMER1  reading value from position 9 is 70
CONSUMER1  reading value from position 10 is 44
```



```
 CONSUMER2  reading value from position 0 is 25
 CONSUMER2  reading value from position 1 is 77
 CONSUMER2  reading value from position 2 is 13
 CONSUMER2  reading value from position 3 is 91
 CONSUMER2  reading value from position 4 is 4
 CONSUMER2  reading value from position 5 is 62
 CONSUMER2  reading value from position 6 is 97
 CONSUMER2  reading value from position 7 is 48
 CONSUMER2  reading value from position 8 is 30
```

iv. Discussion:

While working on this IPC I learned how semaphore makes sure that only one process can access the numbers, even though both of the consumer1 and consumer2 were started at the same time. Threads are a good way while working with multiple processes because it ensures synchronization.

4. IPC Method: Sockets   Language: Python   OS: Windows.

i. Description of the Design:

To run the Sockets method two separate terminals, need to be open. First start the producer.py which will establish the connection and then run the consumer.py. I used the python's built in socket library for this method. After this I used the system call socket( ) to build the socket. Then I used s.bind(Host , Port) system call to connect the host and port. Then I used s.listen ( ) to allow the server to listen to the request of the client. I sued s.accept ( ) to accept the address of the client. I used conn.recv (4096) to receive any data from that port .Another system call used was conn.send(data) to actually send the data to the client.

On the Consumer.py I used socket ( ) to create a socket. After this I used s.connect (Host, Port) to establish the connection of server and client. I used array to store the 100 random integers. Then s.send (data) was used to send the data.

ii. Source Code:

(Consumer.py)

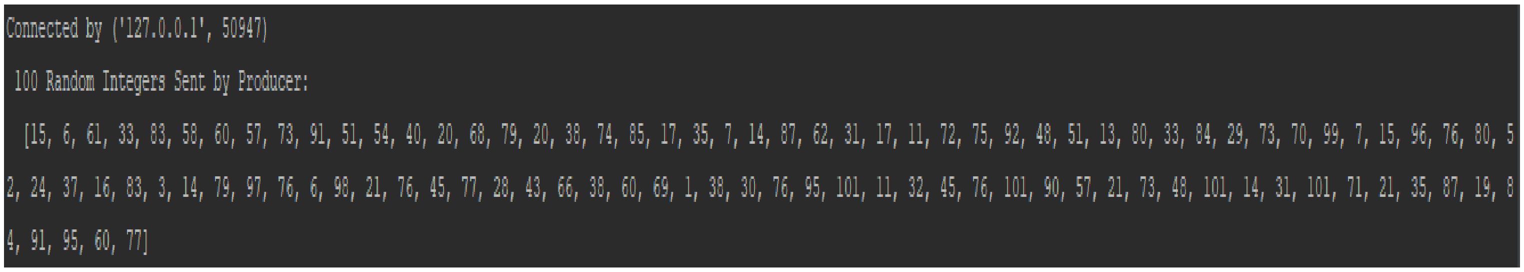
```
import socket, pickle, random

HOST = 'localhost'
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

arr = ([random.randint(1,101), random.randint(1,101) ,random.randint(1,101),
       random.randint(1,101),random.randint(1,101)
       ,random.randint(1,101),random.randint(1,101),
       random.randint(1,101),random.randint(1,101),random.randint(1,101),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01)

,random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,
101),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),random.randint(1,101),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01), random.randint(1,101),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101)

,random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,
101),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01)

,random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,
101),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),random.randint(1,101),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),

random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,101),random.randint(1,1
01),
random.randint(1,101),random.randint(1,101)])


data_string = pickle.dumps(arr)
s.send(data_string)

data = s.recv(4096)
data_arr = pickle.loads(data)
s.close()
print ('100 Random Integers Received by Consumer: \n', repr(data_arr))
```

(Producer.py)

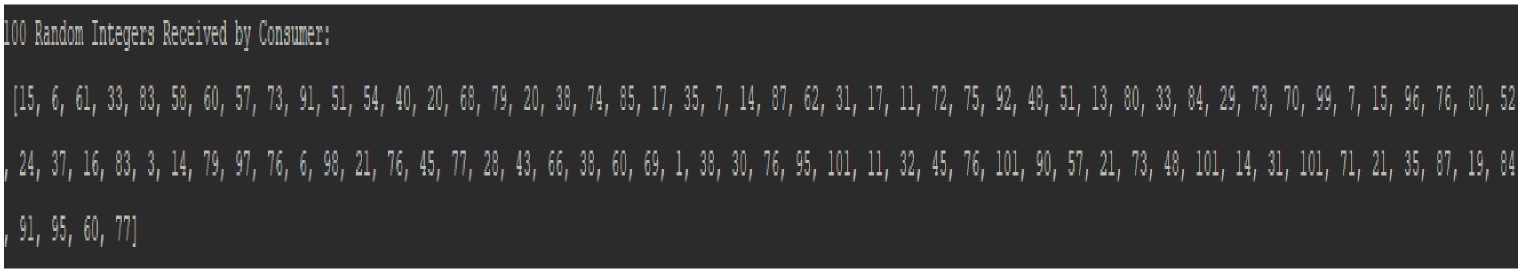
```
import socket, pickle

HOST = 'localhost'
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print ('Connected by', addr)
while 1:
   data = conn.recv(4096)
   if not data: break
   conn.send(data)
   data_arr = pickle.loads(data)
```

```
conn.close()
print(" 100 Random Integers Sent by Producer: \n ",repr(data_arr))
```

iii. Result:



```
Connected by ('127.0.0.1', 50947)

 100 Random Integers Sent by Producer:

  [15, 6, 61, 33, 83, 58, 60, 57, 73, 91, 51, 54, 40, 20, 68, 79, 20, 38, 74, 85, 17, 35, 7, 14, 87, 62, 31, 17, 11, 72, 75, 92, 48, 51, 13, 80, 33, 84, 29, 73, 70, 99, 7, 15, 96, 76, 80, 5
2, 24, 37, 16, 83, 3, 14, 79, 97, 76, 6, 98, 21, 76, 45, 77, 28, 43, 66, 38, 60, 69, 1, 38, 30, 76, 95, 101, 11, 32, 45, 76, 101, 90, 57, 21, 73, 48, 101, 14, 31, 101, 71, 21, 35, 87, 19, 8
4, 91, 95, 60, 77]
```



```
100 Random Integers Received by Consumer:

 [15, 6, 61, 33, 83, 58, 60, 57, 73, 91, 51, 54, 40, 20, 68, 79, 20, 38, 74, 85, 17, 35, 7, 14, 87, 62, 31, 17, 11, 72, 75, 92, 48, 51, 13, 80, 33, 84, 29, 73, 70, 99, 7, 15, 96, 76, 80, 52
, 24, 37, 16, 83, 3, 14, 79, 97, 76, 6, 98, 21, 76, 45, 77, 28, 43, 66, 38, 60, 69, 1, 38, 30, 76, 95, 101, 11, 32, 45, 76, 101, 90, 57, 21, 73, 48, 101, 14, 31, 101, 71, 21, 35, 87, 19, 84
, 91, 95, 60, 77]
```

iv. Discussion:

Using python to implement socket I learned a lot. I first tried doing this method on java but it seemed very complicated. After looking at an example from the internet on python I found it much simpler. The connection between server and client can be established with fewer lines of code in python. I was facing trouble generating the random numbers and storing it in the array with python. Therefore, I had to call random.randint 100 times inside the array to make it work.