RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT - 2

Course Code	19CSC304A
Course Name	Operating Systems
Programme	B. Tech
Department	CSE
Faculty	FET

Name of the Student	K Srikanth
Reg. No	17ETCS002124
Semester/Year	5th Semester/ 3rd Year
Course Leader/s	Ms. Jishmi Jos Choondal

Declaration Sheet				
Student Name	K Srikanth			
Reg. No	17ETCS002124			
Programme	B. Tech		Semester/Year	5th Semester / 3rd Year
Course Code	19CSC304A			
Course Title	Operating Systems			
Course Date	14/09/2020	to	16/02/2021	
Course Leader	Ms. Jishmi Jos Choondal			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student	K Srikanth	Date	18/01/2021
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. in CSE
Semester/Batch	5th / 2018		
Course Code	19CSC304A	Course Title	Operating Systems
Course Leader(s)	Ms. Jishmi Jos Choondal/Ms. Naveeta		

Assignment			
Register No.	17ETCS002124	Name of Student	K Srikanth

Sections		**Marking Scheme**	**Max Marks**	**First Examiner Marks**	**Moderator Marks**
Question 1	Q1.1	Introduction to 16-bit, 32- bit or 64-bit operating systems	01		
	Q1.2	Reasons for the transition from 16-bit to 32- bit and 32-bit to 64-bit operating systems	01		
	Q1.3	Reasons for the transition from 64-bit to 128-bit operating systems	02		
	A1.4	Stance with justification	01		
		Part A	**05**		
estion 2	Q2.1	Introduction to NRU, FIFO, LRU and second chance algorithms	02		
	Q2.2	Compute the page replaced on a page fault	08		

estion3		**B2 Max Marks**	**10**		
	Q3.1	Problem solving approach for spooler	02		
	Q3.2	Design and implementation of spooler	06		
	Q3.3	Results and analysis of spooler	02		
		B2 Max Marks	**10**		
		Total Assignment Marks	**25**		

Course Marks Tabulation				
Component- 1(B) Assignment	**First Examiner**	**Remarks**	**Moderator**	**Remarks**
Q1				
Q2				
Q3				
Marks (out of 25)				
Signature of First Examiner				Signature of Second Examiner

Question 1

1.1)

Introduction

- **16-bit Operating System**

  16-bit Operating System were firstly developed back in year 1978 when intel i**ntroduced its 8086** Processer which was capable of handling a $\mathbf{2^{16}} = \mathbf{65,536}\ \boldsymbol{bits}$ of data transferring and would hold the **maximum main memory of 64 Kilobytes**. These Operating System used to have **16 Bit and 8 Bit Registers to Handle most of the work load.** HP and Intel were the first on maybe who introduced this Processers to do mathematical and computational work and the Operating System didn't have a Graphic interface to interact with it was just **Command Line Interface** and most of consumers were not interested in buying computer until **32-bit Operating System** as they thought that Computers were only for people who work in science field.

- **32- bit Operating System**

  Then comes the Era where It was the turning point on how people looked at computer Introduction of 32-bit Operating System in the year 1984 when **intel introduced its 80386 Processer** which was capable of handling a $\mathbf{2^{32}} = \mathbf{4,29,49,67,296}\ \boldsymbol{bits}$ of data transferring and would hold the **maximum main memory of 4 Gigabytes**. These Operating Systems used to have a **32-Bit ,16 Bit and 8 Bit Registers to Handle most of the work load**. Apple and Microsoft Designed their Operating System which would support a 32-Bit Processer in 1986 right after intel announced their Micro Processer but compared to **16 Bit Operating System 32-bit Operating System** was provided with a **Graphic User Interface** which led the boost of computing to next stage cause many of the consumers felt using a **Graphic User Interface was easier then interacting with Command Line Interface**

- **64-bit Operating System**

  The Present Era where we live in the world where it's all 64-bit Operating System which allowed us to push the Computer Science into huge leap forward. This started basic in **early 2000's** when **AMD64 was released in 2002** and by that time Linux was the first Operating System to support both **x86-64 Bit Computing** and then **came Windows 96 and Mac OS** were the first among to make use of 64-bit Operating System and Processer as it was capable of handling a $\mathbf{2^{64}} = \mathbf{1.844674407E19}\ \boldsymbol{bits}$ of data transferring and would hold the **maximum main memory of 16 Exabytes**. These Operating Systems used to have a **64-Bit**,**32-Bit ,16 Bit and 8 Bit Registers to**

Handle most of the work load. Which are being used to process larger amount of data and computer simulation.

1.2)

The Primary Reasons for the Transition from 16-bit to 32- bit and 32-bit to 64-bit operating systems were

- **Data Bus**

This is one of the primary reasons why the transition happened as we know that,

16-bit Operating Systems were capable of handing $\mathbf{2^{16}} = \mathbf{65,536}\ \boldsymbol{bits}$ at a single instance of time

32-bit Operating Systems were capable of handing $\mathbf{2^{32}} = \mathbf{4,29,49,67,296}\ \boldsymbol{bits}$ at a single instance of time

64-bit Operating Systems are capable of handing $\mathbf{2^{64}} = \mathbf{1.844674407}\boldsymbol{E}\mathbf{19}\ \boldsymbol{bits}$ at a single instance of time

As we can see that there was not sufficient amount of data transfer memory when you compare **16-bit and 32-bit Operating Systems with 64-bit Operating Systems** and for next few years that should do it so that was one of the reasons why the transition happened

- **Performance**

Now coming to Performance this would the also be one of the primary reasons why the transition happened as we know that,

16-bit Operating Systems were capable of handing $\mathbf{2^{16}} = \mathbf{64\ Kilobytes}$ of main memory

32-bit Operating Systems were capable of handing $\mathbf{2^{32}} = \mathbf{4\ Gigabytes}$ of main memory

64-bit Operating Systems are capable of handing $\mathbf{2^{64}} = \mathbf{16\ Exabytes}$ of main memory

As we know to for page replacement, we need a buffer which can handle lot of indexes to store or fetch from Secondary Memory and when you compare **16-bit and 32-bit Operating Systems with 64-bit Operating Systems I don't think that we are using 16 Exabytes of RAM anywhere in the world but it's kind of like theory vs Practical.**

1.3.)

128-bit transition can happen it's not like it won't happen but let's consider the same points that I've written in why transition happened

- **Data Bus**

  16-bit Operating Systems were capable of handing $\mathbf{2^{16}} = \mathbf{65,536}\ \boldsymbol{bits}$ and 32-bit Operating Systems were capable of handing $\mathbf{2^{32}} = \mathbf{4,29,49,67,296}\ \boldsymbol{bits}$ and 64-bit Operating Systems are capable of handing $\mathbf{2^{64}} = \mathbf{1.844674407}\boldsymbol{E}\mathbf{19}\ \boldsymbol{bits}$ at a single instance of time so in the near future let's say that we need more amount of data that has to be capable of handling for all the simulations that we are doing now in the field of science then definitely we need a 128-bit register which would handle 128-bit Operating System for processing more amount of data which would be $\mathbf{2^{128}}$ **= 3.402823669E38 bits** But that would just be a overkill for Home Computers**.**

- **Performance**

  16-bit Operating Systems were capable of handing $\mathbf{2^{16}} = \mathbf{64\ Kilobytes}$ and 32-bit Operating Systems were capable of handing $\mathbf{2^{32}} = \mathbf{4\ Gigabytes}$ and 64-bit Operating Systems are capable of handing $\mathbf{2^{64}} = \mathbf{16\ Exabytes}$ of main memory I think when you look at 128-bit Operating System in case of main memory we would never consume the entire thing and we don't have that amount of Resources for $\mathbf{2^{128}}$ **Bytes of RAM.** When you look at 64-Bit Operating System we didn't use the full potential of a 64-Bit Architecture So when you look at Performance Wise it would definitely be overkill in field of Computer Science and Home Computers.

1.4)

128-Bit Operating System will be Happening mostly under 10 Years but I don't see its full potential from any field of Computer Science but if it happens then I think supercomputers are the one that would be using it. See the processors already exist. And they've existed for decades, too. So do 256-bit and 512-bit processors. Problem is that they don't really have a generic-purpose role in modern computers as it's like using a large air hammer to put a small nail in your wall to hang a picture It's just overkill.

Question 2

2.1)

Introduction

- **NRU algorithm**

It is one of the page replacement algorithms. This algorithm removes a page at random from the lowest numbered non-empty class. When a page needs to be replaced, the Operating System divides pages into 4 classes.

Class 0 : Not Referenced, Not Modified

Class 1 : Not Referenced, Modified

Class 2 : Referenced, Not Modified

Class 3 : Referenced, Modified

Example:

Page	Time of Last Reference (in clock ticks)	Referenced Bit	Modified Bit
0	**200**	**0**	**0**
1	**295**	**0**	**0**

Now looking at the Example we can see that there were no operations performed on **Page 0 and Page 2** so based on Time of Last Reference we can see that the **Page 0** was the one that was **Not Recently Used** so we Replace that Page.

- **FIFO algorithm**

It is one of the page replacement algorithms. The operating system keeps track of all pages in the memory in a queue the least used page will be in front of the queue the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for replacement.

Example :

Consider page reference,

1	2	3	4	5	6

Find Page that will be removed from the queue and the reference string is sorted using Loaded Time in an incremental order so Page 1 would be the one that would be replaced as it was the first one to arrive in the queue

- **LRU algorithm**

It is one of the page replacement algorithms. Least Recently Used (LRU) algorithm is approach of Greedy algorithm where the page to be replaced when its least recently used.

Example :

Consider page reference,

2	3	1	7	0	1

Find Page that will be removed from the queue and the reference string is sorted using Time of Last Reference in an Decremental order so Page 2 would be the one that would be replaced as it was the least used one to in the list

- **Second chance algorithm**

It is one of the page replacement algorithms. The Second Chance page replacement algorithm, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between the time will not be replaced. The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.

Condition: Check if the First Element in the list has a reference bit “0” if so, then make it “1” and pop that element and add it at the end of the list else if the reference bit “1” then replace the page

Example :

Consider page reference,

2	3	1	7	0	1

Find Page that will be removed from the queue and the reference string is sorted using Time of Last Reference and wasn't referred in an Decremental order so Page 2 would be the one that would be replaced as it was the one that wasn't used and Reference Bit was "0".

2.2)

Given,

Table 1

Page	Loaded Time (in clock ticks)	Time of Last Reference (in clock ticks)	R	M
0	250	280	0	1
1	120	285	1	0
2	265	282	0	0
3	110	295	1	0
4	185	289	1	1
5	135	283	0	0
6	275	291	1	1
7	115	279	1	0

To find out which page is supposed to be replaces using

- **NRU Algorithm (Not Recently Used)**

Given,

Page	Time of Last Reference (in clock ticks)	Referenced Bit	Modified Bit
0	280	0	1
1	285	1	0
2	282	0	0
3	295	1	0
4	289	1	1
5	283	0	0
6	291	1	1
7	279	1	0

Now we sort the table out using Time of Last Reference in an Decremental order.

Page	Time of Last Reference (in clock ticks)	Referenced Bit	Modified Bit
3	295	1	0
6	291	1	1
4	289	1	1
1	285	1	0
5	283	0	0
2	282	0	0
0	280	0	1
7	279	1	1

Now that we are sorted with our table using Time of Last Reference in an Decremental order. Looking at the Time of Last Reference, Referenced Bit and Modified Bit using **NRU Algorithm (Not Recently Used). Page 2** has to be the one that has to be replaced.

Page to be Replaced -> Page 2 using NRU

Scenario 1: In this Scenario we look for a page based on Time of Last Reference and let's say that it's not even referenced or modified at any time then we can replace the page
Example:

Page	**Time of Last Reference (in clock ticks)**	**Referenced Bit**	**Modified Bit**
0	**200**	**0**	**0**
1	**295**	**0**	**0**

Now looking at the Example we can see that there were no operations performed on **Page 0 and Page 2** so based on Time of Last Reference we can see that the **Page 0** was the one that was **Not Recently Used** so we Replace that Page.

Scenario 2: In this Scenario we look for a page based on Time of Last Reference and let's say that it's was referenced once and never been modified at any time then we can replace that particular page
Example:

Page	**Time of Last Reference (in clock ticks)**	**Referenced Bit**	**Modified Bit**
0	**295**	**1**	**1**
1	**295**	**1**	**0**

Now looking at the Example we can see that there was one operation perform on the Page 1 and never modified also we can see that both **Page 0 and Page 1** were referred at the same time then looking the **R and M bits** we can replace that particular page

Scenario 3:

In this Scenario we look for a page based on Time of Last Reference and let's say that it's was never been referenced once and has been modified once at any time then we can replace that particular page

Example:

Page	Time of Last Reference (in clock ticks)	Referenced Bit	Modified Bit
0	**295**	**0**	**1**
1	**295**	**1**	**1**

Now looking at the Example we can see that there was one operation perform on the **Page 1** and was never been referenced once also we can see that both **Page 0 and Page 1** were referred at the same time then looking the **R and M bits,** we can replace that particular page

Scenario 4:

In this Scenario we look for a page based on Time of Last Reference and let's say that it's was been referenced once and has been modified once at any time then we can replace that particular page

Example:

Page	Time of Last Reference (in clock ticks)	Referenced Bit	Modified Bit
0	**295**	**1**	**1**
1	**292**	**1**	**1**

Now looking at the Example we can see that there were operations performed on the **Page 1** and **Page 0 and Page 1** was the page that was **Not Recently Used** by looking the **R and M bits,** we can replace that particular page

- **FIFO Algorithm (First in First Out)**

Given,

Page	Loaded Time (in clock ticks)
0	250
1	120
2	265
3	110
4	185
5	135
6	275
7	115

Now we sort the table out using Loaded Time in an incremental order.

Page	Loaded Time (in clock ticks)
3	110
7	115
1	120
5	135
4	185
0	250
2	265
6	275

Now that our table is sorted. Using the FIFO Algorithm (First in First Out) the Page Number 3 is the one that came in first according to the loaded time so it has to be the one that has to be replaced.

Page to be Replaced -> Page 3 using FIFO

- **LRU Algorithm (Least Recently Used)**

Given,

Page	**Time of Last Reference (in clock ticks)**
0	280
1	285
2	282
3	295
4	289
5	283
6	291
7	279

Now we sort the table out using Time of Last Reference in an Decremental order.

Page	**Time of Last Reference (in clock ticks)**
3	295
6	291
4	289
1	285
5	283
2	282
0	280
7	279

- Now that our table is sorted. Using the **LRU Algorithm (Least Recently Used)** the Page Number 7 is the one that was **Least Recently Used** according to **Time of Last Reference** so it has to be the one that has to be replaced.

Page to be Replaced -> Page 7 using LRU

- **Second Chance Algorithm**

Given,

Page	Time of Last Reference (in clock ticks)	Referenced Bit
0	280	0
1	285	1
2	282	0
3	295	1
4	289	1
5	283	0
6	291	1
7	279	1

Now we sort the table out using Time of Last Reference in an Decremental order.

Page	Time of Last Reference (in clock ticks)	Referenced Bit
3	295	1
6	291	1
4	289	1
1	285	1
5	283	0
2	282	0
0	280	0
7	279	1

Now that we are sorted the table now, we can find the page that has to be replaced

Condition: Check if the First Element in the list has a reference bit “0” if so, then make it “1” and pop that element and add it at the end of the list else if the reference bit “1” then replace the page

3	6	4	1	5	2	0	7

Now that we have our list ready let's check the condition one by one

Step 1:

3	6	4	1	5	2	0	7

Let's check the first element in the list and see if the reference bit is equal to "0" or "1"
So, For Page 3 the reference Bit is "1" so we pop the element from the list and add it at the end and update the table.

Page 3 Popped

6	4	1	5	2	0	7	3

Updated Table

Page	Time of Last Reference (in clock ticks)	Referenced Bit
3	295	0
6	291	1
4	289	1
1	285	1
5	283	0
2	282	0
0	280	0
7	279	1

Step 2:

6	4	1	5	2	0	7	3

Let's check the first element in the list and see if the reference bit is equal to "0" or "1"
So, For Page 6 the reference Bit is "1" so we pop the element from the list and add it at the end and update the table.

Page 6 Popped

4	**1**	**5**	**2**	**0**	**7**	**3**	**6**

Updated Table

Page	Time of Last Reference (in clock ticks)	Referenced Bit
3	295	0
6	291	0
4	289	1
1	285	1
5	283	0
2	282	0
0	280	0
7	279	1

Step 3:

4	**1**	**5**	**2**	**0**	**7**	**3**	**6**

Let's check the first element in the list and see if the reference bit is equal to "0" or "1"
So, For Page 4 the reference Bit is "1" so we pop the element from the list and add it at the end and update the table.

Page 4 Popped

1	**5**	**2**	**0**	**7**	**3**	**6**	**4**

Updated Table

Page	Time of Last Reference (in clock ticks)	Referenced Bit
3	295	0

6	291	0
4	289	0
1	285	1
5	283	0
2	282	0
0	280	0
7	279	1

Step 4:

1	**5**	**2**	**0**	**7**	**3**	**6**	**4**

Let's check the first element in the list and see if the reference bit is equal to "0" or "1"
So, For Page 1 the reference Bit is "1" so we pop the element from the list and add it at the end and update the table.

Page 1 Popped

5	**2**	**0**	**7**	**3**	**6**	**4**	**1**

Updated Table

Page	**Time of Last Reference (in clock ticks)**	**Referenced Bit**
3	295	0
6	291	0
4	289	0
1	285	0
5	283	0
2	282	0
0	280	0
7	279	1

Step 5:

5	**2**	**0**	**7**	**3**	**6**	**4**	**1**

Let's check the first element in the list and see if the reference bit is equal to "0" or "1"
So, For Page 5 the reference Bit is "0" so we replace the Page and update the table

5	**2**	**0**	**7**	**3**	**6**	**4**	**1**

Using the **Second Chance Algorithm** the Page Number 5 is the one that has to be replaced.

Page to be Replaced -> Page 5 using Second Chance Algorithm

Question 3

3.1) Introduction

Printer Spooler Problem is similar to Producer and Consumer Problem Approach where we have an item buffer which is stores the produced items and consumed items and follows FIFO order similarly the Printer Spooler Problem follows FIFO order where the Consumer is the Printer and Producer is the User with buffer being the Printer Buffer.

Algorithm

1. Start
2. Declaring Lock variables
3. Initializing Buffer Count Variables
4. Declaring Buffer Array

User Function

1. Start
2. For Loops begins
   I. Generation of random process numbers
  II. Mutex Entry Lock
 III. Add the Process to the Buffer
  IV. Increment the Buffer_in Count
   V. Write "Process Added"
  VI. Mutex Exit Lock
3. Loop End
4. End

Printer Function

1. Start
2. For Loop begins
   I. Declaration of Process Variable
  II. Mutex Entry Lock
 III. Print the Process from the Buffer
  IV. Increment the Buffer_out Count
   V. Write "Process Printed"
  VI. Mutex Exit Lock
3. Loop End
4. End

Main Function

5. Declaration of Buffer_size variable
6. Read Buffer_size
7. Set Buffer_size for the Buffer Array
8. Create Threads for User and Printer Functions
9. Join Threads for User and Printer Functions
10. Stop

Flowchart

Main Function

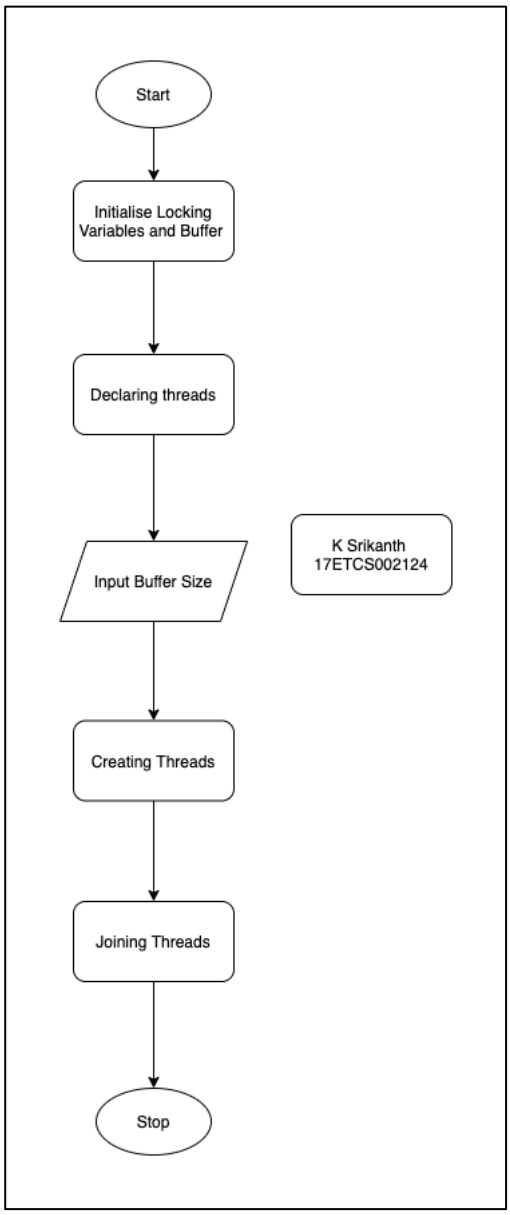

Figure 1 Main Function Flowchart for Printer Spooler Problem

User and Printer Functions

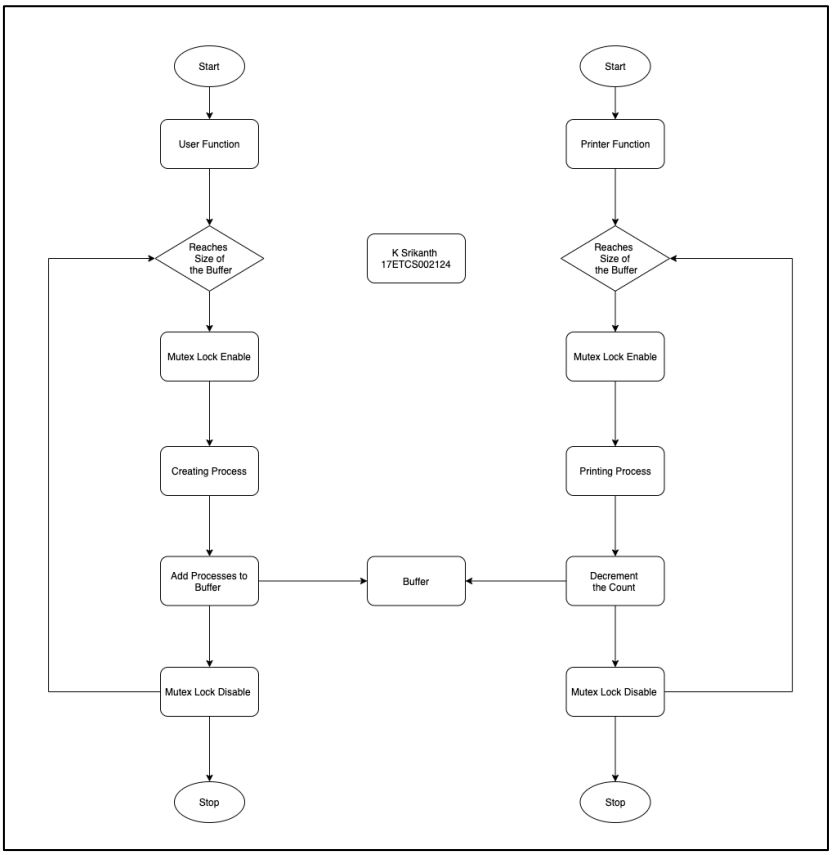

Figure 2 User and Printer Function Flowchart for Printer Spooler Problem

3.2)

C++ Code

Declaration

```cpp
#include <iostream>
#include <thread>
#include <cstdlib>
#include <unistd.h>
#include <semaphore.h>
using namespace std;

sem_t mutex_var, empty, full;
int buffer_in = 0, buffer_out = 0;
int *buffer;
```

Figure 3 C++ Program for Printer Spooler

User Function

```cpp
void *User_function(int buff_size) {
    /*
    ********** K Srikanth 17ETCS002124 **********
    This Function is to create random user process
    for the printer spooler so it can execute it one after one
    */
    for (int i = 0; i < buff_size ; i++) {
        int process = 1 + (rand() % 1000000);
        // Entry Lock Code
        sem_wait(&empty);
        sem_wait(&mutex_var);

        //  Critical Region Code
        buffer[buffer_in] = process;
        cout << endl;
        cout << "User Process : PID-" <<process << endl;
        buffer_in = (buffer_in + 1);

        // Exit Lock Code
        sem_post(&mutex_var);
        sem_post(&full);
    }
    cout << endl;
}

```

Figure 4 C++ Program for Printer Spooler (User Function)

Printer Function

```cpp
void *Printer_function(int buff_size) {
    /*
    ********** K Srikanth 17ETCS002124 **********
    This Function is to print the user process from the
    printer spooler and complete the process one by one
    */

    for (int i = 0; i < buff_size; i++) {
        int process;
        // Entry Lock Code
        sem_wait(&full);
        sem_wait(&mutex_var);

        //  Critical Region Code
        process = buffer[buffer_out];
        buffer_out = (buffer_out + 1) ;
        cout << endl;
        cout<< "Printed the Process :  PID-" << process << endl;

        // Exit Lock Code
        sem_post(&mutex_var);
        sem_post(&empty);
}
}
```

Figure 5 C++ Program for Printer Spooler (Printer Function)

Main Function

```
62  int main() {
63      int buff_size;
64      cout << "********** K Srikanth 17ETCS002124 **********" << endl;
65      cout << endl;
66      cout << "Enter the Printer Spooler Size : " <<;
67      cin >> buff_size;
68      buffer = new int[buff_size];
69      sem_init(&mutex_var, 0, 1);
70      sem_init(&empty, 0, buff_size);
71      sem_init(&full, 0, 0);
72      cout << "********** User Printing Queue **********" << endl;
73      thread t1(User_function,buff_size);
74      sleep(1);
75      cout<<"********** Now Printing **********" <<endl;
76      thread t2(Printer_function,buff_size);
77      sleep(1);
78      t1.join();
79      t2.join();
80      return 0;
81  }
```

Figure 6 C++ Program for Printer Spooler (Main Function)

3.3)

Result

To compile the code

```
g++ -std=c++14 -o filename filename.cpp
```

Then you I will get an executable file and to run the file

Windows

```
./a.out
```

Linux / Mac

```
. /filename
```

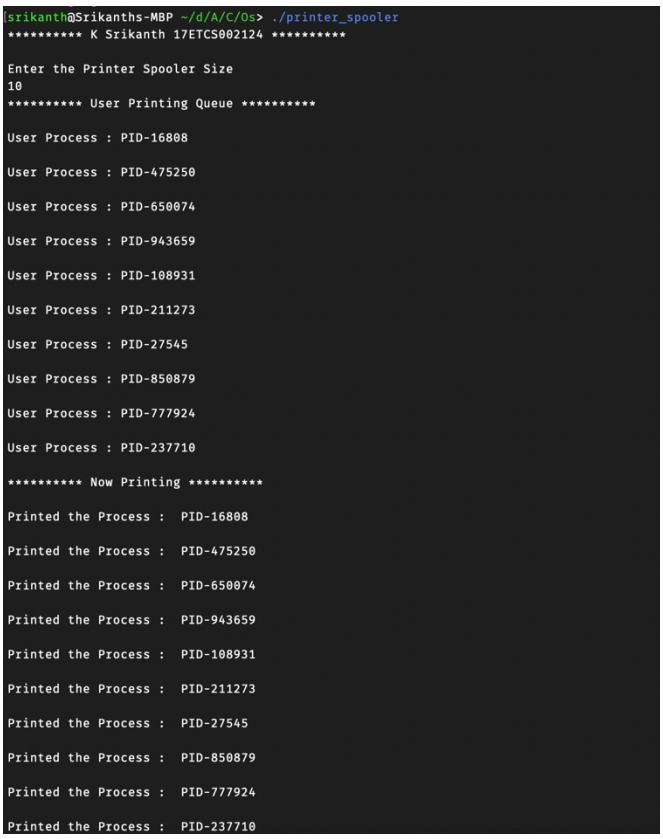

Figure 7 C++ Program Output for Printer Spooler Problem

Conclusion

Printer Spooler Problem is Real life implementation of Producer and Consumer Problem where we have an item buffer which is stores the produced items and consumed items and follows FIFO order similarly the Printer Spooler Problem follows FIFO order where the Printer takes one by one document from the Spooler when the User creates multiple process to print.