

ASSIGNMENT

Course Code 19CSC304A
Course Name Operating Systems
Programme B. Tech
Department Computer Science and Engineering
Faculty Engineering and Technology

Name of the Student Deepak R
Reg. No 18ETCS002041
Semester/Year 5th/2020
Course Leader/s Ms. Jishmi Jos Choondal

Declaration Sheet			
Student Name	Deepak R		
Reg. No	18ETCS002041		
Programme	B. Tech	Semester/Year	5 th /2020
Course Code	19CSC301A		
Course Title	Operating Systems		
Course Date		to	
Course Leader	Ms. Jishmi Jos Choondal		
Declaration <p>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.</p>			
Signature of the Student		Date	
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	5 th / 2018		
Course Code	19CSC304A	Course Title	Operating Systems
Course Leader(s)	Ms. Jishmi Jos Choondal		

Assignment					
Register No.		18ETCS002041	Name of Student		Deepak R
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		
Question 2	Q2.1	Introduction to NRU, FIFO, LRU and second chance algorithms	02		
	Q2.2	Compute the page replaced on a page fault	08		
		B2 Max Marks	10		

Question 3	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)				
<div>Signature of First Examiner</div> <div>Signature of Second Examiner</div>				

Solution for Question 1

Introduction to 16-bit, 32-bit or 64-bit operating systems

16-bit OS

Some of the earliest processors like the 8088 (1979) and Intel 286 (1982) had a capacity of 16-bits. This meant that each register could only store about 65,536 memory addresses. The 8088 could access only 1 MB of data.

The computer would have had to break down larger values into small components. It would have taken a fairly long time for the computer to collect and assimilate these pieces, hence slowing down the speed of execution.

16-bit colour refers to the number of colours that can be displayed — that is, 2^{16} or 65,536 colours can be displayed. Most Windows 95 applications were 16-bit.

32-bit OS

The first single-chip 32-bit microprocessor was HP FOCUS that was launched in 1981. Until 2008, 32-bit systems were the standard.

In general, a 32-bit system can access up to 4GB of RAM but no more. Very few systems today use a 32-bit architecture. Those which still do are termed X86 systems. This includes the Classic Mac OS, OS X, Windows 95, 98, XP, and Vista.

A 32-bit program is compatible with a 64-bit operating system, but a 32-bit OS cannot run a 64-bit program. The 64-bit versions of Windows have two different install folders, one of which is the 32-bit directory.

64-bit OS

A 64-bit processor can handle large amounts of data at once. A single 64-bit register can store 18,446,744,073,709,551,616 values at once. This is about 4 billion times the capacity of a 32-bit processor.

A 64-bit processor could theoretically access well over 17 billion GB or 16 exabytes of memory. This is several million times more than required by an average computer, but this large value facilitates improved performance.

64-bit systems allocate memory blocks a lot more efficiently than 32-bit systems. This prevents the eventual slowing down of the system. The user can multitask on their computer without the fear of doing too much for the processor to handle.

Reasons for the transition from 16-bit to 32-bit and 32-bit to 64-bit operating systems

16-bit to 32-bit operating systems

32-Bit integer calculations are obviously well suited to a 32-bit processor. Occasions where 32-bit integers are required include graphics and manipulation of large data structures. While a 16-bit processor can simulate 32-bit arithmetic using double-precision operands, 32-bit processors are much more efficient.

While 16-bit processors can use segment registers to access more than 64K elements of memory, this technique becomes awkward and slow if it must be used frequently. A program that must continually change the segment register to access data structures (especially single data structures that are bigger than 64K in size) can waste a considerable amount of time computing segment values. Even

worse, since the addresses that must be manipulated when computing data record locations that are greater than 16 bits wide, address computations are also slower because of all the double-precision math involved. A 32-bit processor can offer a linear 32-bit address space with accompanying quick address calculations on a 32-bit data path.

32-bit to 64-bit operating systems

32-bit OS can address a maximum of 4 GB (4,294,967,296 bytes) of RAM whereas 64 bit OS can handle more data than 32 bit OS, in numbers, it can address 264 memory addresses, i.e actually 18-Quintillion GB of RAM.

A system with a 64-bit processor can run either of a 64-bit or 32-bit version of an operating system installed. On the other hand systems with a 32-bit processor could run 32 bit OS properly but 64-bit OS would not run at its full capability.

A 64-bit processor is more recommended for multitasking and other heavy application execution due to its high performance as compared to a 32-bit processor.

Reasons for the transition from 64-bit to 128-bit operating systems

While there are currently no mainstream general-purpose processors built to operate on 128-bit *integers* or addresses, a number of processors do have specialized ways to operate on 128-bit chunks of data. The IBM System/370 could be considered the first simple 128-bit computer, as it used 128-bit floating-point registers.

Regarding computing, 128-bit units are commonplace nowadays for SIMD instructions. Recent Intel CPU are now well beyond 128-bit, e.g. with AVX-512 which processed data in 512-bit chunks. So in that sense, any operating system that can properly save and restore the context of an application using AVX-512 is already a "512-bit operating system".

Similarly, long-term storage has long gone beyond 64-bit. ZFS is probably the most prominent and I guess one of the earliest filesystems that can deal with 128-bit files and volumes. So any operating system that supports ZFS (which is to say most modern operating systems) supports 128-bit filesystems.

Stance with justification

We have already moved into an n-bit era of computing. Computer operating systems can efficiently support computation at any number of bits. Memory and storage can enumerate all the storage and memory on the planet in a unique way thanks to the internet. There will never be another bit width driven industry wide change to our operating systems.

There are still conditions where specialized handling of a 128-bits could be done more quickly or more economically with specialized hardware, and our current CPUs and our display cards already do that. But we don't need new operating systems because the current operating systems already contain all the abstractions needed to deal with these special cases.

Solution for Question 2

Introduction to NRU, FIFO, LRU and second chance algorithms

NRU - The Not-Recently-Used Page Replacement Algorithm

When a process starts, all its page entries are marked as not in memory (i.e. the present/absent bit). When a page is referenced a page fault will occur. The R (reference) bit is set and the page table entry modified to point to the correct page. In addition the page is set to *read only*. If the page is later written to, the M (modified) bit is set and the page is changed so that it is *read/write*.

Updating the flags in this way allows a simple paging algorithm to be built.

When a process is started up all R and M bits are cleared (set to zero). Periodically (e.g. on each clock interrupt) the R bit is cleared. This allows us to recognise which pages have been recently referenced.

When a page fault occurs (so that a page needs to be evicted), the pages are inspected and divided into four categories based on their R and M bits.

Class 0 : Not Referenced, Not Modified

Class 1 : Not Referenced, Modified

Class 2 : Referenced, Not Modified

Class 3 : Referenced, Modified

The Not-Recently-Used (NRU) algorithm removes a page at random from the lowest numbered class that has entries in it. Therefore, pages which have not been referenced or modified are removed in preference to those that have not been referenced but have been modified

The First-In, First-Out (FIFO) Page Replacement Algorithm

This algorithm simply maintains the pages as a linked list, with new pages being added to the end of the list. When a page fault occurs, the page at the head of the list (the oldest page) is evicted. Whilst simple to understand and implement a simple FIFO algorithm does not lead to good performance as a heavily used page is just as likely to be evicted as a lightly used page.

LRU - least recently used algorithm

In *least recently used* algorithm when a page fault generates, we choose the page that was used very less in the past. In other words, optimal algorithm checks forward reference while LRU checks backward reference in the page reference string.

The page table entry records the time when the page was last referenced and it is modified every time when the page is referenced. It is basically works on the assumption that if a page is not used frequently in the past then chances are there that it might not be required in the future also.

The Second Chance Page Replacement Algorithm

The second chance (SC) algorithm is a modification of the FIFO algorithm. When a page fault occurs the page at the front of the linked list is inspected. If it has not been referenced (i.e. its reference bit is clear), it is evicted. If its reference bit is set, then it is placed at the end of the linked list and its reference bit reset. The next page is then inspected. In this way, a page that has been referenced will be given a second chance.

In the worst case, SC operates in the same way as FIFO. Take the situation where the linked list consists of pages which all have their reference bit set. The first page, call it a, is inspected and placed at the end of the list, after having its R bit cleared. The other pages all receive the same treatment. Eventually page a reaches the head of the list and is evicted as its reference bit is now clear. Therefore, even when all pages in a list have their reference bit set, the algorithm will always terminate.

Solution for Q2.2 Compute the page replaced on a page fault

NRU - The Not-Recently-Used Page Replacement Algorithm

When a process is started up all R and M bits are cleared (set to zero). Periodically (e.g. on each clock interrupt) the R bit is cleared. This allows us to recognise which pages have been recently referenced. When a page fault occurs (so that a page needs to be evicted), the pages are inspected and divided into four categories based on their R and M bits.

Pages

R M (0,0):- 2, 5

R M (0,1):- 0

R M (1,0):- 1, 3, 7

R M (1,1):- 4, 6

In this R M bit 0,0 is selected = 2 and 5 are from NOT MODIFIED AND NOT REFERENCED, In them any one can be taken randomly I took 2.

Page 2 or 5 is replaced by using NRU .Page 2 is taken randomly by NRU from 2 and 5.

The First-In, First-Out (FIFO) Page Replacement Algorithm

This algorithm simply maintains the pages as a linked list, with new pages being added to the end of the list. When a page fault occurs, the page at the head of the list (the oldest page) is evicted.

Order according to LEAST LOAD TIME

Page. Load time

3. 110

7. 115

1. 120

5. 135

4. 185

0. 250

2. 265

6. 275

In this page 3 has lowest loaded time so it is replaced first then according to order above they are replaced.

LRU - *least recently used* algorithm

Order in Least time of reference order

Page 7. 0. 2. 5. 1 4. 6. 3

Last time of Reference 279. 280. 282. 283. 285. 289. 291. 295

In this 7 has least last time of reference so it is used last at 279 clock cycle seconds. So it's used long time ago so it as the most possibility for not being used in the future also so 7 is replaced in page fault by LRU.

The Second Chance Page Replacement Algorithm

First we have to arrange in FIFO ORDER means in least loaded time order.

From FIFO. 3. 7. 1. 5. 4. 0. 2. 6

Then have to check whether "R" BIT IS ZERO or not If R bit Is 1 Then we have to place that page in last and check for check next .If R Bit is zero then that page should be replaced. "R" bit 1 means it is most referenced most means it will be used again and again . So it will be given a second chance.

Check for ~~3.~~ 7. 1. 5. 4. 0. 2. 6 3 has R bit 1 so it placed last and given 2nd chance heck for

Next. 7. 1. 5. 4. 0. 2. 6 3 7 has R bit 1 so it is placed last and given 2nd CHANCE and check
for Next 1. 5. 4. 0. 2. 6 3 7 1 has R bit 1 so it is placed last and given 2nd chance check for
next

 5. 4. 0. 2. 6 3 7 1 5 has R bit. 0 so this page is replaced in page fault. So this
means it is not referenced so it is not used most. So 5 is removed by Second chance

Solution for Question 3

Printer spooling happens on the computer that is attached to the printer or on the network server that handles printing. Print jobs are sent to the spooler program and the program then sends those documents one at a time to the printer in the order they were received. While documents are waiting in line to be printed, you can continue to work on your computer because all of the print functions are being handled in the background by the spooler.

print jobs are sent to a print spool before being transmitted to the printer. For example, when you print a document from within an application, the document data is spooled to a temporary storage area while the printer warms up. As soon as the printer is ready to print the document, the data is sent from the spool to the printer and the document is printed.

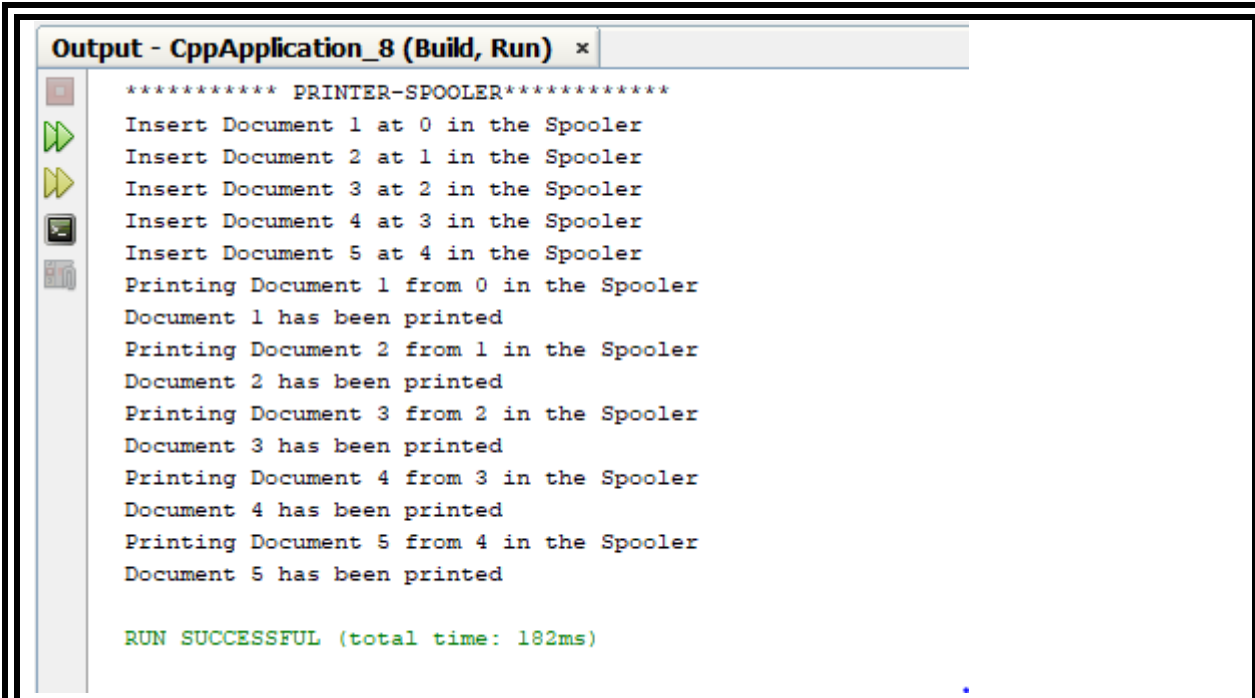
Algorithm for spooler

```
spooler() {  
  
    for(int i=0;i<BufferSize;i++){  
  
        sem_wait(&empty);  
  
        buffer[in] = doc[i];  
  
        Display("Inserting Document %d at %d in the Spooler\n",buffer[in],in);  
  
        in = (in+1)%BufferSize;  
  
        sem_post(&full);  }}  
  
printer(){  
  
    for(int i=0;i<BufferSize;i++){  
  
        sem_wait(&full);  
  
        doc[i] = buffer[out];  
  
        Display("Printing Document %d from %d in the Spooler\n",doc[i], out);  
  
        Display("Document %d has been printed\n",doc[i]);  
  
        out = (out+1)%BufferSize;  
  
        sem_post(&empty)  }}
```

Program Source code

```
1 //program done By Despak R 18ETCS002041
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <pthread.h>
6 #include <semaphore.h>
7 #define MaxItems 5 //MAX ITEMS PRODUCER CAN PRODUCE AND CONSUMER CAN CONSUME
8 #define BufferSize 5
9
10 sem_t empty,full; //SEMAPHORE VARIABLES
11 int in = 0; //GLOBAL DECLARATION
12 int out = 0;
13 int buffer[BufferSize];
14 int doc[MaxItems] = {1,2,3,4,5};
15
16 void *spooler(void *args) {
17     for(int i=0;i<BufferSize;i++){
18         sem_wait(&empty);
19         buffer[in] = doc[i];
20         printf("Inserting Document %d at %d in the Spooler\n",buffer[in],in);
21         in = (in+1)%BufferSize;
22         sem_post(&full);
23     }
24 }
25
26 void *printer(void *args){
27     for(int i=0;i<BufferSize;i++){
28         sem_wait(&full);
29         doc[i] = buffer[out];
30         printf("Printing Document %d from %d in the Spooler\n",doc[i], out);
31         printf("Document %d has been printed\n",doc[i]);
32         out = (out+1)%BufferSize;
33         sem_post(&empty);
34     }
35 }
36
37 int main(int argc, char** argv) {
38     printf("***** PRINTER-SPOOLER*****\n");
39
40     pthread_t thread1,thread2;
41
42     sem_init(&empty,0,BufferSize);
43     sem_init(&full,0,0);
44
45     pthread_create(&thread1,NULL,(void *)spooler,NULL);
46     pthread_create(&thread2,NULL,(void *)printer,NULL);
47
48     pthread_join(thread1,NULL);
49     pthread_join(thread2,NULL);
50
51     sem_destroy(&empty);
52     sem_destroy(&full);
53
54     return (EXIT_SUCCESS);
55 }
56
57
```

Output



```
***** PRINTER-SPOOLER*****
Insert Document 1 at 0 in the Spooler
Insert Document 2 at 1 in the Spooler
Insert Document 3 at 2 in the Spooler
Insert Document 4 at 3 in the Spooler
Insert Document 5 at 4 in the Spooler
Printing Document 1 from 0 in the Spooler
Document 1 has been printed
Printing Document 2 from 1 in the Spooler
Document 2 has been printed
Printing Document 3 from 2 in the Spooler
Document 3 has been printed
Printing Document 4 from 3 in the Spooler
Document 4 has been printed
Printing Document 5 from 4 in the Spooler
Document 5 has been printed

RUN SUCCESSFUL (total time: 182ms)
```

The above is the Result of our Implementation

To test the Spooler, the number of items is very small, due to this the testing is not accurate, there could be a wrong implementation and the program might still work if the thread did not get de-scheduled, hence the simulation should be done for a relatively larger number of items.

This Method Can also be used for generating Banner pages (these are the pages used in computerized printing in order to separate documents from each other and to identify e.g. the originator of the print request by username, an account number or a bin for pickup. Such pages are used in office environments where many people share the small number of available resources).