

|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course Code** | 19CSC304A |
| **Course Name** | Operating Systems |
| **Programme** | B. Tech. |
| **Department** | Computer Science and Engineering |
| **Faculty** | CSE |

#### 

|  |  |
| --- | --- |
| **Name of the Student** | Subhendu Maji |
| **Reg. No** | 18ETCS002121 |
| **Semester/Year** | 5th / 2018 |
| **Course Leader/s** | Ms. Naveeta |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name | Subhendu Maji | | | | | | | |
| Reg. No | 18ETCS002121 | | | | | | | |
| Programme | B. Tech | | | | | Semester/Year | 5th / 2018 | |
| Course Code | 19CSC304A | | | | | | | |
| Course Title | Operating Systems | | | | | | | |
| Course Date |  | | to | |  | | | |
| Course Leader | Ms. Naveeta | | | | | | | |
| **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 | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |

# **Contents**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[**Declaration Sheet** ii](#_Toc61831126)

[**Contents** iii](#_Toc61831127)

[**Marking Scheme** 4](#_Toc61831128)

[**Question No. 1** 6](#_Toc61831129)

[1.1 Introduction to 16-bit, 32- bit or 64-bit operating systems 6](#_Toc61831130)

[1.2 Reasons for the transition from 16-bit to 32- bit and 32-bit to 64-bit operating systems 6](#_Toc61831131)

[1.3 Reasons for the transition from 64-bit to 128-bit operating systems 6](#_Toc61831132)

[1.4 Stance with justification 6](#_Toc61831133)

[**Question No. 2** 7](#_Toc61831134)

[2.1 Introduction to NRU, FIFO, LRU and second chance algorithms 7](#_Toc61831135)

[2.1 Compute the page replaced on a page fault 9](#_Toc61831136)

[**Question No. 3** 10](#_Toc61831137)

[3.1 Problem solving approach for spooler 10](#_Toc61831138)

[3.2 Design and implementation of spooler 11](#_Toc61831139)

[3.3 Results and analysis of spooler 11](#_Toc61831140)

|  |  |  |  |
| --- | --- | --- | --- |
| **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. | | | 18ETCS002121 | Name of Student |  | SUBHENDU MAJI | | |
|  |  | **Marking Scheme** | | |  | |  |  |
|  | 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** | |  |  |
|  | 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** | |  |  |
| 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 No. 1**

**Solution to Question No. 1:**

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

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

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

## 1.4 Stance with justification

# **Question No. 2**

**Solution to Question No. 2:**

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

**Page Fault** – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

## 2.1 Introduction to NRU, FIFO, LRU and second chance algorithms

**NRU- Not Recently Used Page Replacement Algorithm**

This algorithm removes a page at random from the lowest numbered non-empty class. Implicit in this algorithm is that it is better to remove a modified page that has not been referenced in at least one clock tick than a clean page that is in heavy use.

When page is modified, a modified bit is set. When a page needs to be replaced, the Operating System divides pages into 4 classes.

|  |  |  |
| --- | --- | --- |
|  | R | M |
| Class 0 | Not Referenced | Not Modified |
| Class 1 | Not Referenced | Modified |
| Class 2 | Referenced | Not Modified |
| Class 3 | Referenced | Modified |

Out of above 4 categories, NRU will replace a Not Referenced, Not Modified page, if such page exists.

This algorithm implies that a Modified but Not Referenced is less important than a Not Modified and Referenced.

**FIFO- First In First Out Page Replacement Algorithm**

In this algorithm, the operating system keeps track of all pages in the memory in a 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 removal.

**LRU- Least Recently Used Page Replacement Algorithm**

In this algorithm page will be replaced which is least recently used. In other words, LRU replaces the page that has not been used for the longest period of time

**Second Chance Page Replacement Algorithm**

In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations 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.

It can be implemented by adding a “second chance” bit to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process). Thus, a page with the “second chance” bit set to 1 is never replaced during the first consideration and will only be replaced if all the other pages deserve a second chance too.

## 2.1 Compute the page replaced on a page fault

We are given,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

Table 1 Given time of loading, time of last access, and the R and M bits for each page

**NRU** makes an approximation to replace the page based on referenced and modified bits.

Based upon R and M bits, there are 4 classes:

|  |  |  |
| --- | --- | --- |
|  | R | M |
| Class 0 | Not Referenced | Not Modified |
| Class 1 | Not Referenced | Modified |
| Class 2 | Referenced | Not Modified |
| Class 3 | Referenced | Modified |

Table 2 Classes based upon referenced and modified bits

According to the Table 2,

|  |  |
| --- | --- |
| **Page** | Belongs to |
| 0 | Class 1 |
| 1 | Class 2 |
| 2 | Class 0 |
| 3 | Class 2 |
| 4 | Class 3 |
| 5 | Class 0 |
| 6 | Class 3 |
| 7 | Class 2 |

NRU algorithm removes class 0 pages first. Hence, **Page 2** and **Page 5** will be replaced by NRU.

**FIFO** replacement algorithm works in a way that it will replace the page that arrived the earliest.

According to Table 1,

FIFO queue will be as follows:



Page 3 got the least value of Loaded Time, which means it arrived the earliest.

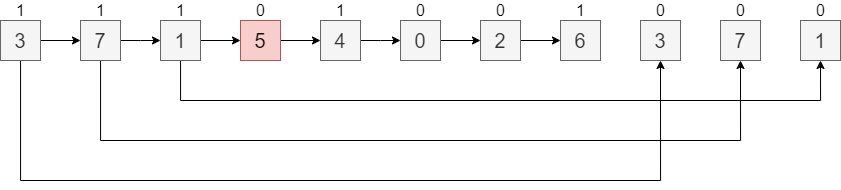
Hence, **Page 3** will be replaced by FIFO.

**LRU** is a page replacement algorithm that replaces the page that is least recently used. Thus, LRU takes decision on the basis of the Last Reference and page having least Last Reference is being replaced.

According to Table 1, Page 7 got the least Time of last Reference.

Hence, **Page 7** will be replaced by LRU.

**Second chance** algorithm works by adding a 'second chance' bit to each memory frame whenever page is referenced associated to that frame, this bit is set and when replacement is needed, this bit is reset (set to 0).



Hence, **Page 5** will be replaced by Second Chance Algorithm.

# **Question No. 3**

**Solution to Question No. 3:**

## 3.1 Problem solving approach for spooler

This problem is a classic Producer-Consumer Problem.

The producer and consumer share a fixed-size buffer used as a queue. The producer’s job is to generate data and put this in the buffer. The consumer’s job is to consume the data from this buffer, one at a time.

## 3.2 Design and implementation of spooler

We can use Semaphore and Mutex to solve our problem.

Data structures we are going to use:

* Header file semaphore.h

We will use semaphore of type sem\_t

Some inbuilt method we need:

|  |  |
| --- | --- |
| **commands** | **purpose** |
| sem\_init | Initializes the semaphore to some initial value |
| sem\_wait | Blocks the calling process until one of its child processes exits or a signal is received. |
| sem\_post | Generates interrupt that is sent to a process |
| sem\_destroy | Destroys the semaphore to avoid memory leak |

* Header file pthread.h

We will use mutex of type pthread\_mutex\_t

|  |  |
| --- | --- |
| **commands** | **purpose** |
| pthread\_mutex\_init | Initializes the mutex |
| pthread\_mutex\_lock() | Blocks the calling process until one of its child processes exits or a signal is received. |
| pthread\_mutex\_unlock() | Generates interrupt that is sent to a process |
| pthread\_mutex\_destroy() | Destroys the mutex to avoid memory leak |

## 3.3 Results and analysis of spooler

**Bibliography**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. <https://shivammitra.com/c/producer-consumer-problem-in-c/#>
2. <https://hackmd.io/@25077667/os-hw4#fn1>