

**BACHELOR OF COMPUTING SCIENCE CSC1007 OPERATING SYSTEM TRIMESTER 2, YEAR 2021/22**

**Group Assignment Team Project: FINAL REPORT**

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Student ID** |
| **1** | **ABDULLAH RAHMAN** | **2100972** |
| **2** | **LIM KEI YIANG** | **2100593** |
| **3** | **SHYAM PRASAD** | **2100756** |
| **4** | **IZZ DANIAL** | **2100735** |

# Table Of Content

Contents

[Table Of Content 2](#_Toc99835656)

[1. Message Device Driver & Message Client 3](#_Toc99835657)

[Overview 3](#_Toc99835658)

[Limitations 3](#_Toc99835659)

[Output 3](#_Toc99835660)

[Source code 4](#_Toc99835661)

[MessageDev.c (Loadable Kernel Module) 4](#_Toc99835662)

[Flow Chart for Kernel 12](#_Toc99835663)

[2.1 LinkedList.c 13](#_Toc99835664)

[Purpose 13](#_Toc99835665)

[Breakdown of functions 13](#_Toc99835666)

[1. Insert (LinkedList \*list, int value) 13](#_Toc99835667)

[2. insertAt(LinkedList \*list, int index, int value) 14](#_Toc99835668)

[3. delete (LinkedList \*list, int index) 15](#_Toc99835669)

[2.2 PageReplacementMain.c 16](#_Toc99835670)

[2.3 First in first out (FIFO) page replacement algorithm 17](#_Toc99835671)

[2.3.1 Code Explanation 17](#_Toc99835672)

[2.4 Optimal page replacement algorithm 19](#_Toc99835673)

[2.4.1 Code Explanation 19](#_Toc99835674)

[2.5 Least recently used (LRU) page replacement algorithm 22](#_Toc99835675)

[2.5.1 Code Explanation 22](#_Toc99835676)

[Flow Chart (Page Replacement) 25](#_Toc99835677)

[Reflections 27](#_Toc99835678)

# Message Device Driver & Message Client

## Overview

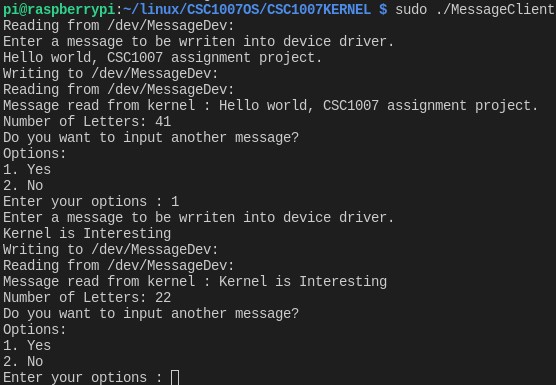
There are two-part to this. One of them is the MessageDev, the loadable kernel module. The other one will be the MessageClient.c which, is the user application that is used to send message to the kernel space and read the message and count from the kernel space.

## Limitations

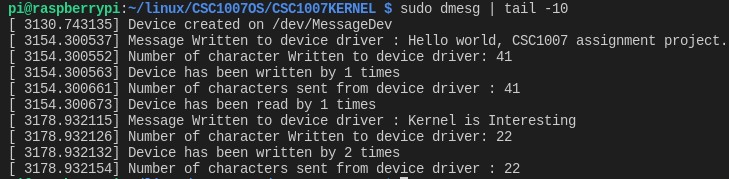
The limitation of the two programs is 200 for the number of characters that can be stored and allowed for input.

## Output

This section will show the result on both the user space application and the device driver. Message Client



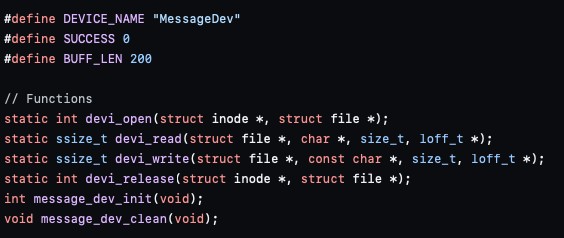
The above shows the flow of the input with some input. Where users will be allowed to choose to input more than one message and have it sent to the kernel space or if they do

Loadable Kernel Module (MessageDev.c)

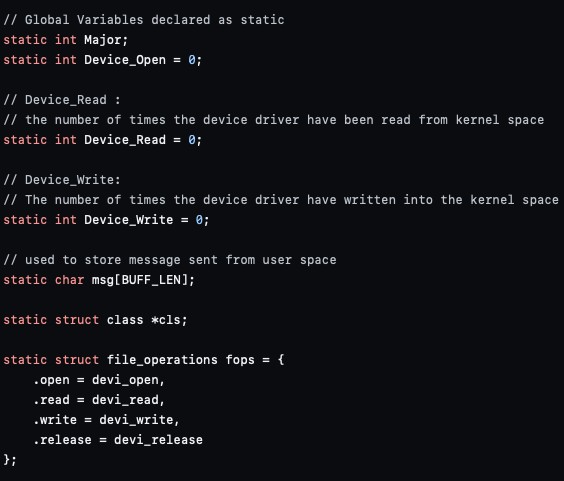
This is the message shown recording the number of times have written to the kernel space or reading from kernel space. It also shows the message have has been written into the kernel space from the userspace.

## Source code

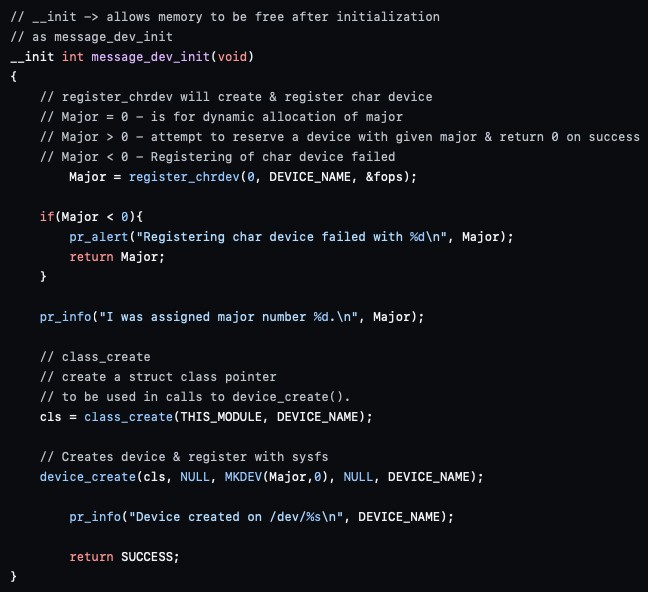
## MessageDev.c (Loadable Kernel Module)



This portion of code above is to declare the functions devi\_open, devi\_read, devi\_release message\_dev\_init and message\_dev\_clean which will be required for the module.



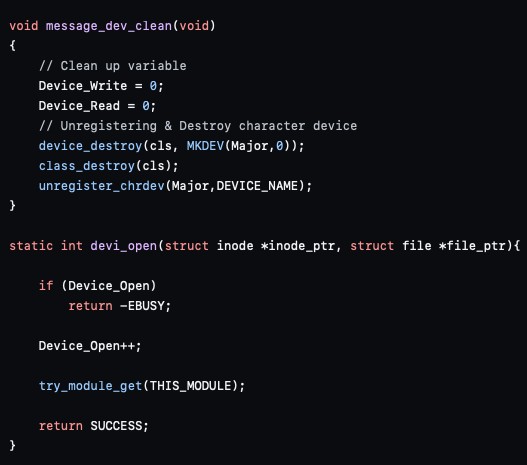
Next, we will declare the variable required. Major, which will be used to store the major number when the device is created. Device\_Open, will be used to record the number of times the device driver has been accessed. Device\_Read is used to record the number of times a read system call has been executed to read the message from the kernel space into the user application. Device\_Write is used to record the number of times a write system call has been executed to call the write operation from user space to kernel space.



Here, we will see the message\_dev\_init function during the initialization phase. It will first register chrdev using the device name we have give and the file\_operations structure.

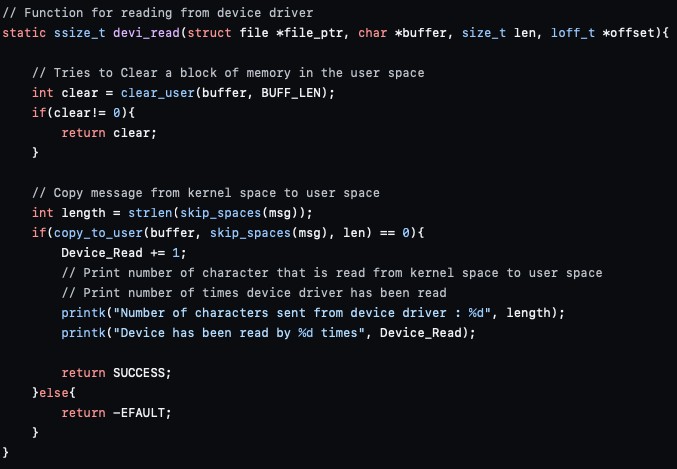
The value 0 defined in the register\_chrdev is for major number is to allow dynamic allocation. Then after, a major number will be assigned. If a major number is less than 0, the program will alert that registering the char device has failed.

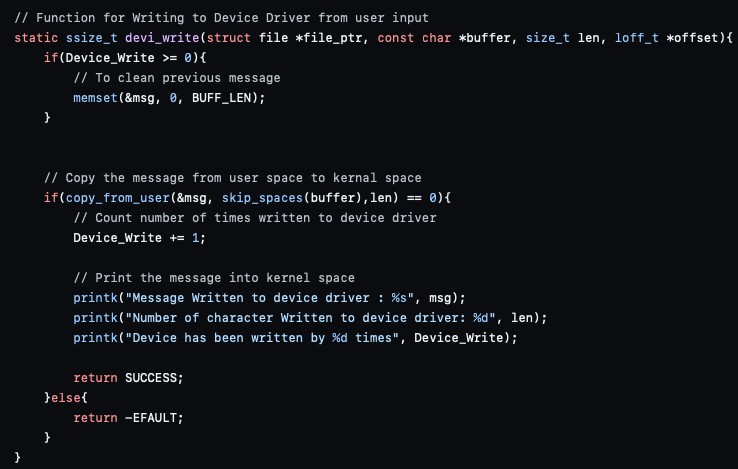
Next, we will create a class structure used for device\_create, which will create a device entry with a specific device name and trigger an event to create a device node.



Message\_dev\_clean function is executed on module exit. As we will need to unregister and destroy device since at the initialization phase we have registered and created a device

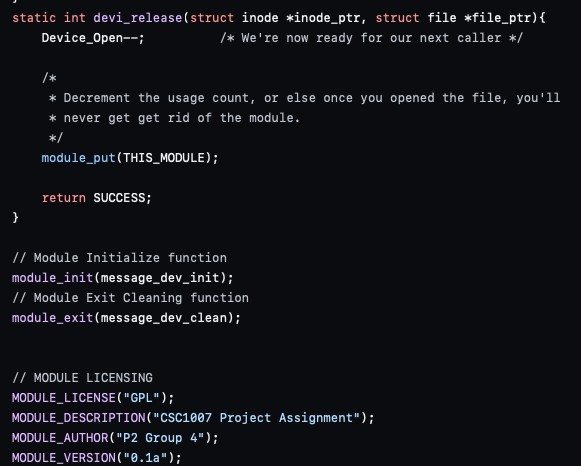
Within devi\_open, it will have device\_open to count the number of time the device has been open and try\_module\_get is to manipulate the module usage count and to protect against removal.





This function devi\_read is called when read system call is executed on the user space application. The purpose of having this function to allow user to get message from kernel space back into the user space and allowing it to be displayed in the user application.

In devi\_write, on the other hand executes when write system call is executed, allowing user to send message from the user application and have it written into the kernel space.

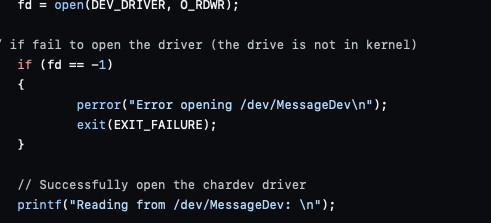


The function devi\_release is called at the last close for the file. Module\_init and module\_exit are used to define the function called on init / exit. Lastly, is that we declare the description of the module is for the project and author being our group p2 group number 4.

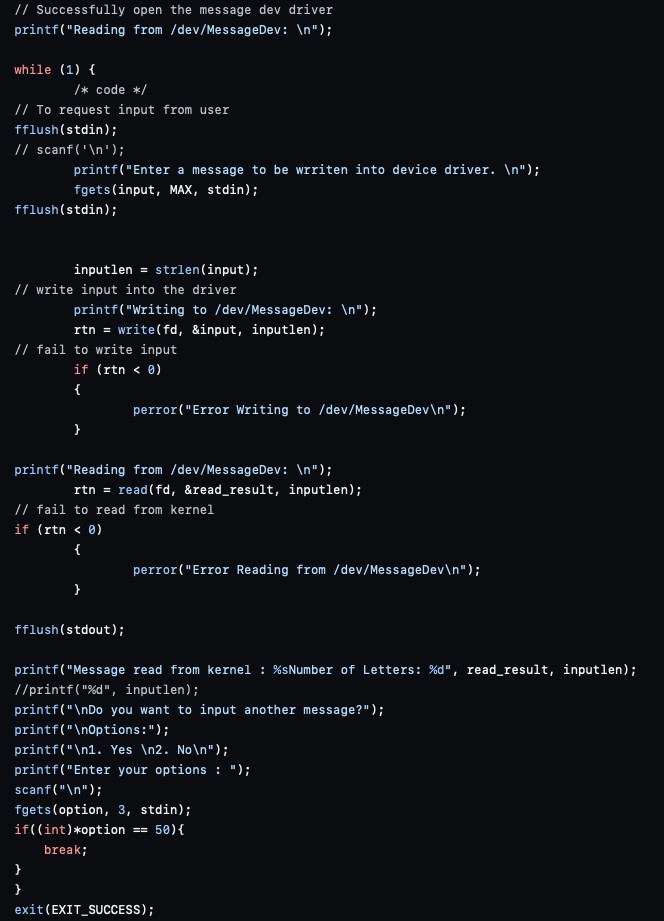
MessageClient.c



We define the device driver location within the user application by assigning it to a static char. We also define a Maximum size for our user input.

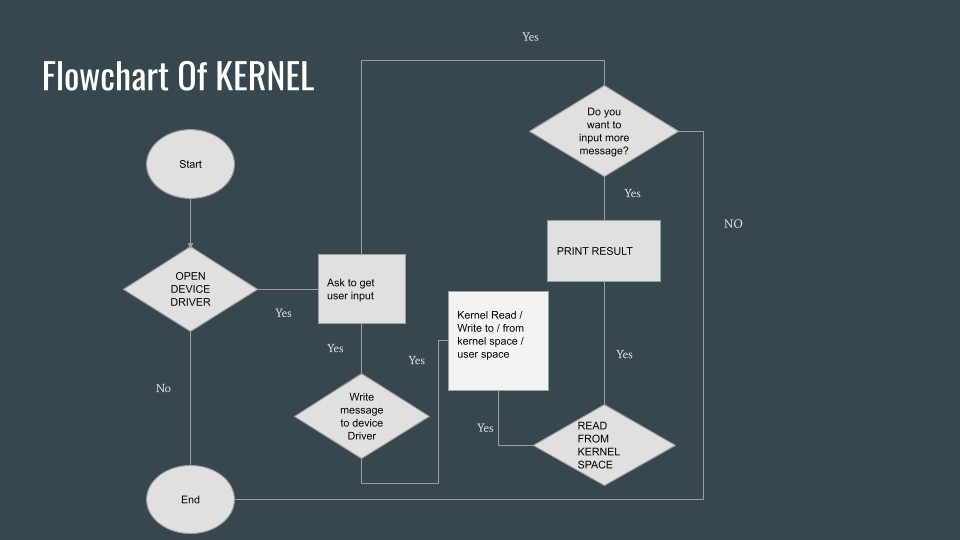


Within this picture, as shown below. We will then attempt to open the device driver with the location defined previously.



After the device driver is successfully open, we will then begin to request user input and when user press enter key. It will then execute write system call to have the message written into the kernel space. Then after, it will execute read system call and print out the message sent and the number of letters of the message. After read system call has been executed, then it will next ask user if they want to continue to enter another message or exit the program.

## Flow Chart for Kernel



# LinkedList.c

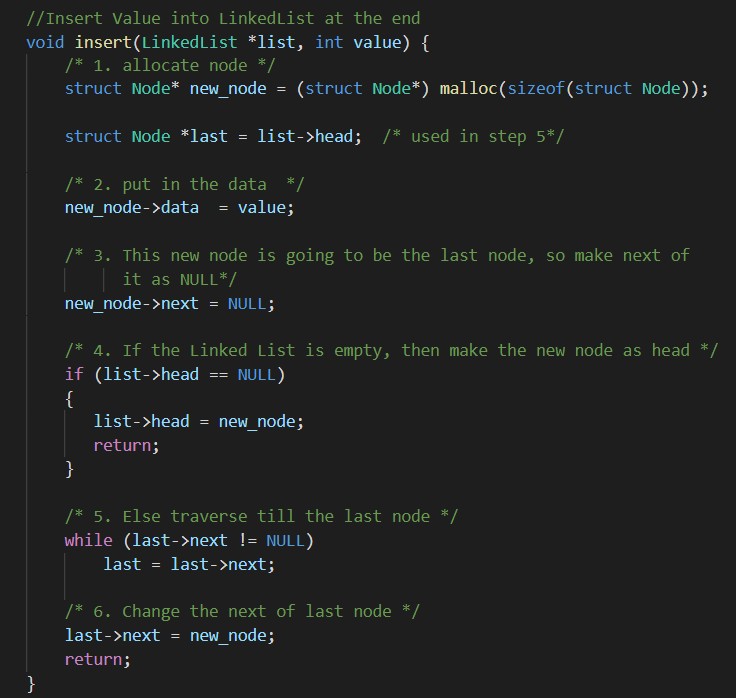
## Purpose

Help aid with implementation of the 3 algorithms. Individual files will be able to access the functions in linkedlist.c to carry out the necessary operations.

## Breakdown of functions

## Insert (LinkedList \*list, int value)

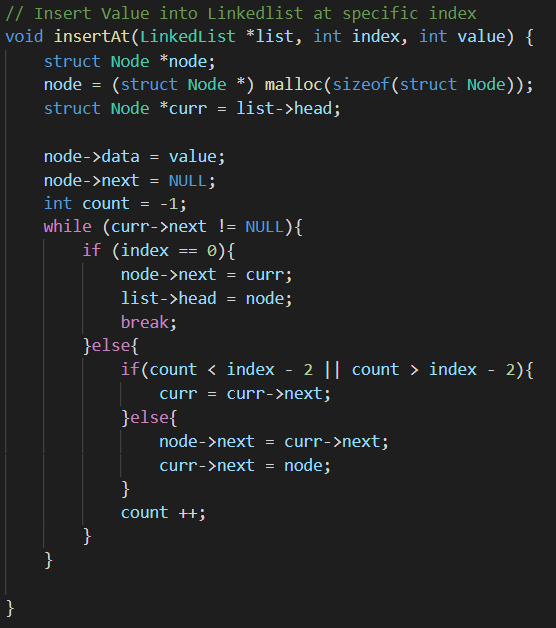
**What it does**: Insert value into the end of the linked list.



1. The function will initially create a new node with a dynamic memory allocation using malloc ().
2. The value passed into the function will be assigned to the new node.
3. Since the new node created will be the last node in the linked list, the next node it points to is set to NULL.
4. The function will then check if the head of the linked list is empty. If it is empty, the new node would be the new head as there are no other values.
5. If linked list is not empty, it will traverse the whole list until the end of the linked list to obtain the last node.
6. The initial, last node of the linked list would then be linked to the new node, effectively assigning it as the new last node.

## insertAt(LinkedList \*list, int index, int value)

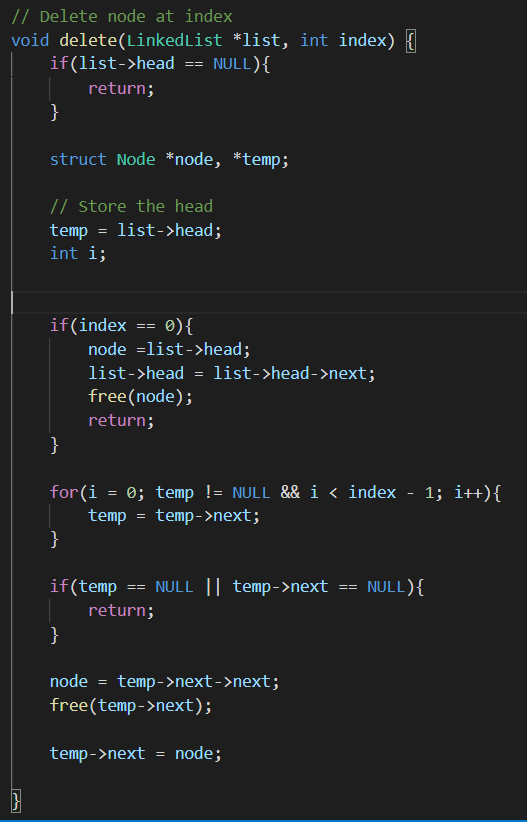
**What it does**: Insert value into specific index of the linked list.



1. The function will initially create a new node with a dynamic memory allocation using malloc ().
2. The value passed into the function will be assigned to the new node.
3. Since the new node created will be the last node in the linked list, the next node it points to is set to NULL.
4. The function will then check if the head of the linked list is empty. If it is empty, the new node would be the new head as there are no other values.
5. If linked list is not empty, it will traverse the whole list as long as index is either more or less than count.
6. If count is the same as the index, we would set the next node that the current index is pointing to as where our new node will point to
7. Then, we will add our node to where the initial index would point to

## delete (LinkedList \*list, int index)

**What it does:** Delete node at specific index of the linked list.



1. The function will check if the list is empty. If the list is empty, it will not delete anything and return
2. If index is 0, it will set the value of the current head to the node, then set the value of the new head as the next value after the head
3. Then we will free the node, meaning that we free the memory that was allocated by malloc() and return
4. if the index is not 0 (done in step 2) or if the list is not empty, and while i remains less than the size of the list, it will traverse the list.
5. once we find the position(the position before the index actually as it is a linked list), we will store the node where the one we want to delete is pointed to
6. we will then free the node (as explained in step 3)
7. we will then point the node before the one that we delete, to the one AFTER the one we deleted, effectively removing the indexed one as the middle man

# PageReplacementMain.c

1. First we will create a char array with the size of MAX that we defined to be 200
2. Then we will create a pointer variable (Reference String/refStr) to the char array that we created
3. Then we will create a refenced linked list with dynamic memory allocation called refll
4. We will then set the head to null and then retrieve the user input for option and number of frames
5. We will then fflush the input the that we retrieved. It will return 0 if it is succesful.
6. We will then loop through the list. Suppose the value of refStr and i added together is between 48 and 57. In that case, we will insert this value into the referenced linked list, and based on the option that the user has selected. It will go into the selected page replacement algorithm.
7. Once the algorithm has finished running, we will then empty the Linked List and free the memory allocated to the referenced linked list

# First in first out (FIFO) page replacement algorithm

Overview

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.

Advantages

It is the easiest algorithm to implement Simplest to understand

Disadvantages

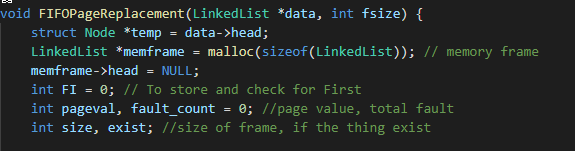
Not as effective as the other page replacement algorithms Every frame needs to be accounted

If a bad replacement, is chosen, page fault will increase massively

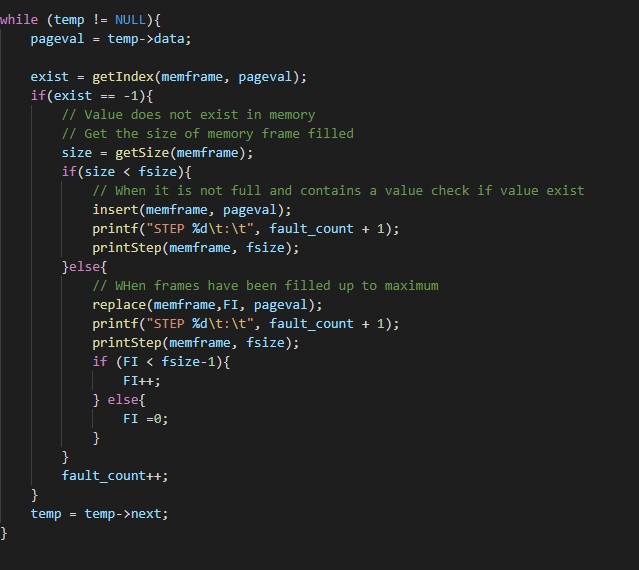
Belady’s Anomaly – A problem that occurs when number of frames is increased. In this situation, more memory is provided to process. Although it should normally result in a reduction in page fault due to the increased frames; However, if FIFO is implemented, the page faults will increase instead

## Code Explanation

2.3.2 Variable Declaration



1. The function will initially create a new node temp and set it as the head
2. It will then create the linked list memframe using dynamic memory allocation
3. Next, it will set the head to null
4. FI is used to store the index to be used for the replace function, page val for page value, fault\_count for the total number of faults
5. size is used to compare the current frame size and the max frame size assigned and exist is to check if the value exists in the memory
   * 1. FIFO Algorithm



1. While user’s input linked list **is not empty**, proceed to next step.
2. If value is in the memory frame, move to the next value in user’s input linked list

Proceed to **step 1**.

1. If value **is not** in the memory frame, the function will check if the memory frame is full If it is **not full**, insert value at next available memory frame Proceed to **step 5.**

**4a)** If memory frame **is full,** replace page at the current index (FI) of memory frame Proceed to **step 4b.**

**4b)** Increase index (FI) count by 1 if it is still lesser than the size of memory frame – 1 If index is bigger than memory frame – 1, set it to 0 Proceed to **step 5.**

**5)** Increase fault\_count Move to the next value in user’s input linked list Repeat this until there are no more values in the linkedlist

# Optimal page replacement algorithm

Overview

In this algorithm, OS replaces the page that will not be used for the longest period in future.

Advantages

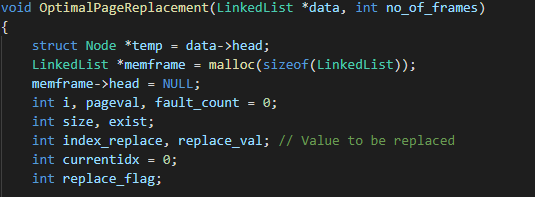
Reduced number of page faults by replacing the page that will least likely be replaced soon.

Disadvantages

Not possible to predict what pages will be used in the future. Time consuming

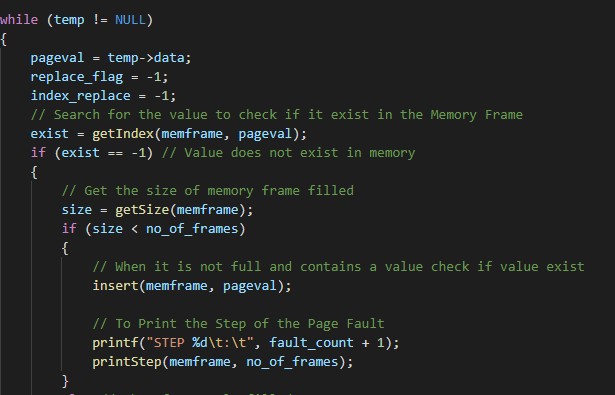
## Code Explanation

* + 1. Variable Declaration



|  |
| --- |
| 1. The function will initially create a new node temp and set it as the head 2. It will then create the linked list memframe using dynamic memory allocation 3. Next, it will set the head to null 4. Head of memory frame is set to null 5. Page value and page fault initialised to 0 6. size is used to compare the current frame size and the max frame size assigned and exist is to check if the value exists in the memory 7. i is used for code(loop) iteration |

Optimal Algorithm



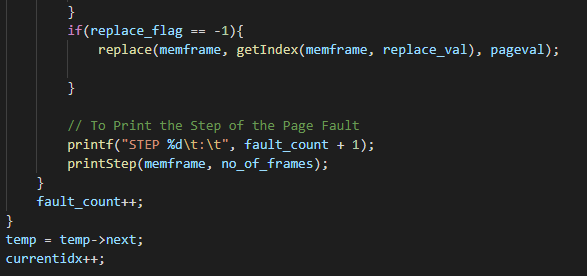
1. While user’s input linked list **is not empty**, set replace\_flag and index\_replace to –1 proceed to next step.
2. If value is in the memory frame, move to the next value in user’s input linked list

Proceed to **step 11**.

1. If value **is not** in the memory frame, the function will check if the memory frame is full If it is **not full**, insert value at next available memory frame Proceed to **step 4.**
2. Increase fault\_count Move to the next value in user’s input linked list Increase currentidx by 1 Proceed to **step 1**



1. If memory frames are full, iterate through the whole memory frame one by one.
2. In every iteration, we will get the value in the memory frame Check if the value exists in the upcoming values of the reference string.
3. If it **does not** exist, replace that specific memory frame with the current value Set replace\_flag to 1 and break the loop Proceed to **step 4.**
4. If it does exist and index\_replace is –1, set index\_replace as index found in future reference string(exist) Set replace\_val as the value in the memory frame with the current index i.
5. If it does exist and index\_replace is not –1, if current index found in future reference string(exist) is more than index\_replace, set index\_replace as current index found Set replace\_val as the value in the memory frame with the current index i.



1. Replace memory frame with the value that has the highest current index found in future reference string(exist) with the new value Proceed to **step 4.**
2. Move to the next value in user’s input linked list Increase currentidx by 1 Proceed to

**step 1**

# Least recently used (LRU) page replacement algorithm

Overview

In this algorithm, OS replaces the page that was least recently used

Advantages

Since the page replaced is the one that is least recently used, Belady’s Anomaly does not

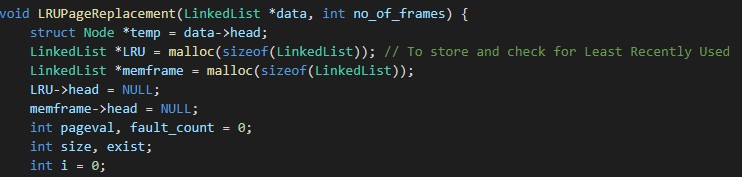
occur It is also easy to choose page which has faulted and hasn’t been used for a long time.

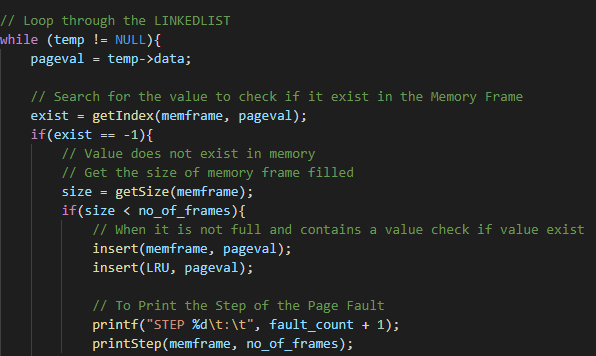
Disadvantages

It requires additional Data Structure to be implemented.

## Code Explanation

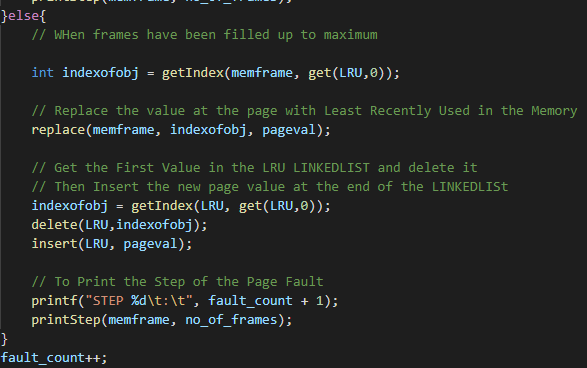
* + 1. Variable Declaration



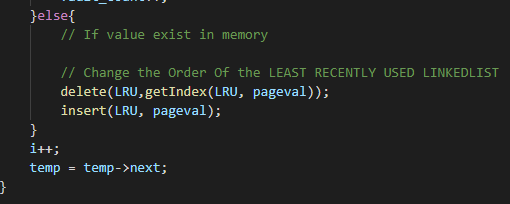
1. The function will initially create a new node temp and set it as the head
2. It will then create the linked list memframe using dynamic memory allocation
3. Next, it will set the head to null
4. Head of memory frame is set to null
5. Page value and page fault initialised to 0
6. size is used to compare the current frame size and the max frame size assigned and exist is to check if the value exists in the memory
7. i is used for code(loop) iteration
   * 1. LRU Algorithm
8. While user’s input linked list **is not empty**, proceed to the next step.
9. If the value is in the memory frame, move to the next value in the user’s input linked

list Proceed to **step 8**.

1. If the value **is not** in the memory frame, the function will check if the memory frame is full If it is **not full**, insert value at next available memory frame and add the value to linked list, LRU Proceed to **step 4.**
2. Increase fault\_count Move to the next value in user’s input linked list Proceed to **step 1.**

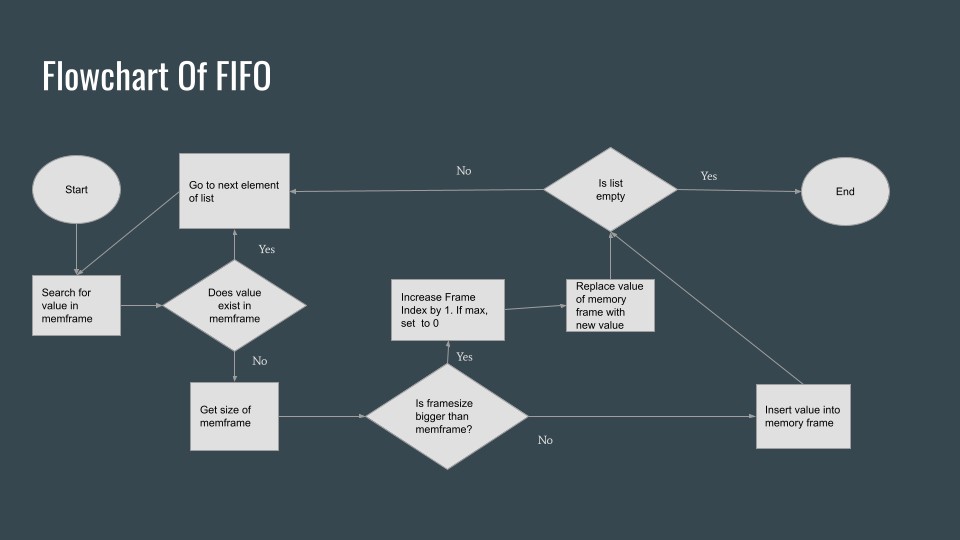


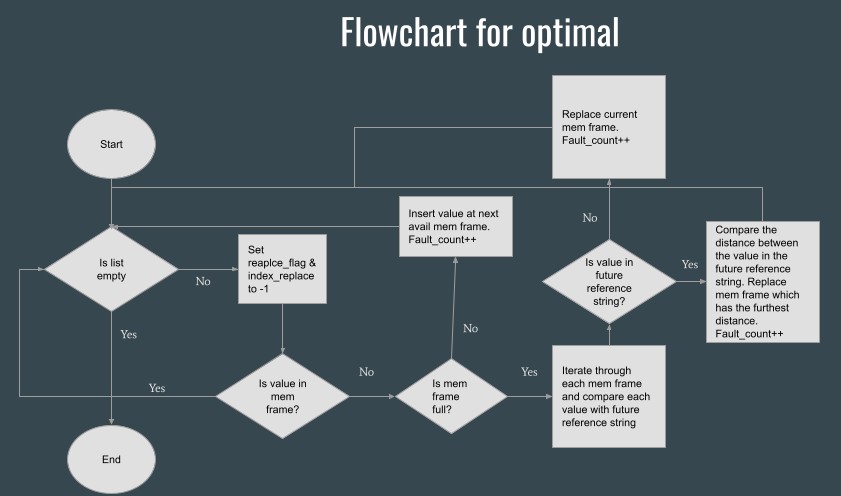
1. If value **is not** in the memory frame and memory frame **is full,** get the index of memory frame that has the page value that was least recently used by checking the first value in LRU linked list.
2. Replace that index in the memory frame with the new value.
3. Get the first value in LRU linked list and delete it Insert new page value at the end of LRU linked list Proceed to **step 4.**

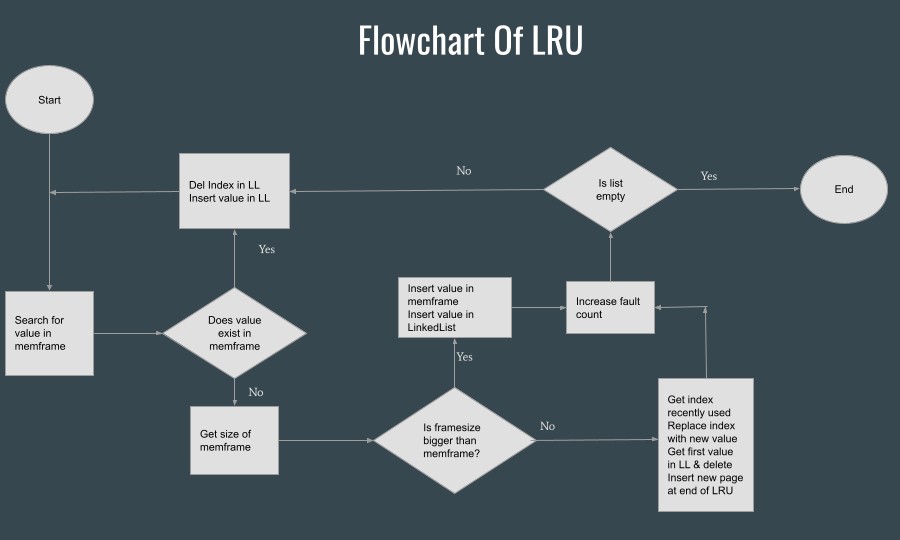


**8)** Find the index in LRU linked list where the value is the same as the current value Delete that index in the linked list Insert the value in the LRU linked list to change its position to the end Proceed to **step 1.**

## Flow Chart (Page Replacement)







# Reflections

LIM KEI YIANG

Throughout the project, I have learned more about writing the loadable kernel module, making use of system calls on the user applications to read and write from the device driver However, throughout the journey of completing this project I have faced many issues as I was not familiarise with what can be used or done when creating the loadable kernel module It took me quite a while to start to understand how the basics of kernel such as what is module\_init and exit and why is there both release and clean Also when working on the kernel module, I also learned that it is not as easy to debug as unlike programming just a basic program there exist a debugging tools that readily available to be used However, for Linux kernel the only way I find issue is through diagnostic messages and each time will have to rmmod to remove the module, make to compile and insmod to install the module over again to test if the issue has been fixed.

Initially, I had thought that the static variable and have been thinking of it as a global variable then throughout the assignment, I then realise that I am wrong as the static is refering to the storage duration (static) Thus, this is at the same time have corrected the way I have think of global variable for loadable kernel module Within the loadable kernel module, I learned that there is different way where we can read data from user space and write it into the kernel space such as get\_user and copy\_from\_user, in such a way where get\_user copies a character/variable at a time However, copy\_from\_user will copy a block of data from user space to the kernel I also have known that I will need to clear the buffer of user space before I send the message back if not there will be leftover messages of previous message in the user space if the previous statement is longer, which is performed using clear\_user() On the user application, I learned how I should use read and write system call as on my first try, I failed to to successfully execute write / read system call without any error But I believe my kernel module is not an ideal one as many other ways can improve the loadable kernel module I have not fully explored the full documentation on the Linux kernel and further understood each project It is also because of this project, in future if I were to work on project that would require me to implement different kind of loadable module.

I will at least already have a basic knowledge of Linux kernel This also let me know the importance of having a loadable kernel module which includes the waste of memory when not in idle and if there are new features or functions they will have to rebuild the base kernel if loadable kernel module is not available.

I am also thankful to my team as work was completed on time.

IZZ DANIAL

My initial impression of this module is that I would have learned about the different type of operating systems such as macOS, windows and Linux This module has given me greater insights on how the operating system works For example, I have learned about the various tasks that takes place in the OS and how it is being executed, when it should be executed and why it is being executed that way I have also learned about the different structures in place in an operating system to ensure that things run properly and avoid encountering into errors and to ensure optimization for the operating system I have also learned how the operating system schedules the processes using different algorithms (FCFS, SJF, SRTF, RR, Priority), different ways to handle deadlocks by either prevention, avoidance or recover from deadlock, multithreading models, process management, process synchronization, file permissions etc The weekly labs and tutorials has helped me gained a better understanding of the concepts as the labs allowed me to have a hands-on try on the concepts taught and the tutorials has provided me with the opportunity to apply the module content This module has given me a greater understanding of operating systems.

This project has taught me more about the different page replacement algorithms in greater detail This project required us to do our own research to figure out how to implement the various algorithms in c language I have also solidified my understanding of the different strengths, weaknesses and limitations of the different algorithms and which algorithms are more practical than the other due to their nature Having to go through the brainstorming process of implementing the algorithm has resulted in me being able to grasp a greater understanding towards the algorithm I have also realised from this group assignment that each individual algorithm has its trade-offs For example, FIFO page replacement algorithm will give the fastest results but at the expense of higher number of page faults For optimal page replacement algorithm, the OS will have lesser number of page faults at the expense of longer time taken to complete the task For least recently used page replacement algorithm, it is efficient and avoids the problem of Belady’s Anomaly but at the expense of the implementation being more expensive, complex and requiring external hardware support This group assignment has also helped me strengthen my c language programming and learn how to tackle problems from a different angle I’ve learned a lot and gained different perspectives from my groupmates such as implementing linked list functions that are reusable to make the code neater.

Lastly, this project work has forged new friendships and I’m very thankful for my group members for constantly rendering help to each other when in need We were able to exchange ideas without having any conflicts when our ideas do not align Overall, I feel that having such group assignments that requires us to search and implement the algorithms used by the operating systems is a good approach to learning the contents of the module as I am able to better absorb and understand the content by engaging in hands on activities.

SHYAM PRASAD

I would like to start my reflection by thanking my group mates for guiding and helping me at every step of the way. Thank you, Kei Yiang, Izz, Abdullah, and Prof Cao Qi.

The main part about Operating Systems that caught my eye is Memory Management and File Systems Normally us normal users, just open a file to use it We don’t think about what is going on in the background Sometimes I wonder why algorithms like Worst-fit algorithm were created I also enjoyed applying things learnt from Data structures in OS Especially using hashing and Linked Lists to solve memory management issues Grouping users into different categories and giving them their access rights was intriguing The hands-on learning given by the labs strengthened my understanding of things learnt during lectures It is also inspiring to see how there are different stages of handing Deadlocks Deadlock detection, deadlock avoidance and deadlock prevention All three levels have different solutions It is inspiring to know how people, just like us, come up with complex algorithms and solutions to solve these issues Translation Lookaside Buffer, TLB, was something I found interesting too It performs as a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval For example, if going into the main memory to retrieve it takes 20ns, TLB might take 10ns If memory exists in TLB, it will exist in Main memory TLB Hit and page fault cannot co-exist.

For this project, I worked on the LRU Page Replacement algorithm.

This project gave me a glimpse into using algorithms to solve real life problems Implementing the algorithms gave me a better and clearer understanding of each of them Identifying their advantages and disadvantages, real-life applications and more Understanding these algorithms in pen and paper was fairly simple But typing them out in code was really challenging for me Sleepless nights with my groupmates trying to slowly debug each line was a painful but a true learning and humbling experience.

This project has enhanced my knowledge in page replacement algorithms, the C language, Linked Lists and pointers This project also taught me how code integration and code standardization is very important It would have been easier to implement all the algorithms using simple arrays but if one person uses Linked List, we should all try to follow suit so as to maintain code readability and code standardization Also, usage of header files and neat writing of code is crucial so that if someone outside reads our code, they will be able to see how the code was integrated They would not need to spend lots of time trying to understand the basic logic flow This enhances the option of code scalability.

Teamwork was exemplified by our use of GIT We used GIT to push and pull, and to stay updated with our teammates' progress.

To conclude, this project gave me a glimpse of working in a real project development team We helped each other at every step of the way, and I am very grateful to my team members for helping and guiding me when necessary.

ABDULLAH

This project has truly taught me many things about various different page replacement algorithms and what it can and is used for in the outside world Like for example, FIFO is an asset-management and valuation method for companies to decide which assets produced or acquired first are sold, used, or disposed of first.

So, although FIFO might be very simplistic and may not seem to be very practical as its just basically a queuing system, it does beg the question of how else FIFO impacts our daily lives and the world surrounding us With that in mind i decided to view what the benefit of FIFO is and why such a simplistic system is still implemented by businesses even when there are other clearly superior options when it comes to faults For businesses, the FIFO method has four key advantages when compared to its competition: Firstly, it is easy to apply, making what would seem to be a drawback be its main appeal and selling point Secondly, the assumed flow of costs will correspond with the normal physical flow of goods which means that income manipulation is almost impossible in this scenario Finally, the balance sheet amount for inventory would also approximate the current market value as the inventory listed would be more recent

With that in mind, FIFO became intrinsically more interesting, and it ignited my interest in how the various other concepts that we have learnt could be of use, even if we don’t pursue an IT job in the future It also does bring up an interesting question of how everything we’ve learnt are linked to one another It's always easy to remain stagnant and just use what we normally have used to do this project by using an array or a list but the challenge that using an unfamiliar concept such as linked list would provide would prove to be beneficial to me and my group It’s about taking a learning opportunity from something as simple as FIFO and making it worthwhile by introducing various restrictions so that I don’t rest on my laurels especially since one of the main requirements for a job in the IT sector is to be able to upgrade your skills and by questioning what you know previously.

Finally, I would also like to thank my groupmates for this wonderful opportunity, and I appreciate how my group covers for each other's shortcomings It was a humbling experience as I realised that in this line of work/education there’s always bound to be someone who’s better than you in some respects and weaker than you in others and what would take me or another groupmate hours to do would be seen as simple for another person Of course, there would always be a feeling of inferiority but at the end of the day, it's up to me as a person to learn from the person or just be content with what I have and remain complacent Although, it's almost humanising to see someone you look up to face

adversity and overcome it through sheer willpower and hard work, it does make me wonder if I have been too complacent with what I have acquired