Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: PAGE REPLACEMENT TECHNIQUES

Aim/Objective:

Efficiently allocate and manage page replacement algorithms and resources to optimize system performance and facilitate the execution of processes and applications. Paging is a memory management technique that allows non-contiguous memory allocations to process and avoids the problem of external fragmentation. Most modern operating systems use paging. With paging, the physical memory is divided into frames, and the virtual address space of a process is divided into logical pages. Memory is allocatedat the granularity of pages. When a process requires a new page, the OS finds a free frame to map this page into and inserts a mapping between the page number and frame number in the page table of the process.

Description:

FIFO (First In First Out) is the simplest page replacement algorithm. With FIFO, logical pages assigned to processes are placed in a FIFO queue. New pages are added at the tail. When a new page must be mapped, the page at the head of the queue is evicted. This way, the page brought into the memory earliest is swapped out. However, this oldest page may be a piece of code that is heavily used, and may cause a page fault very soon, so FIFO does not always make good choices, and may have a higher than optimal page fault rate. The FIFO policy also suffers from Belady's anomaly, i.e., the page fault rate may not be monotonically decreasing with the total available memory, which would have been the expectation with a sane page replacement algorithm. (Because FIFO doesn't care for the popularity of pages, it may so happen that some physical memory sizes lead you to replace a heavily used page, while some don't, resulting in the anomaly.) The FIFO is the simplest page replacement algorithm, the idea behind this is "Replace a page that page is the oldest page of all the pages of the main memory" or "Replace the page that has beenin memory longest".

What is the optimal page replacement algorithm? One must ideally replace a page that will not be used for the longest time in the future. However, since one cannot look into the future, this optimal algorithm cannot be realized in practice. The optimal page replacement has the lowest page fault rate of all algorithms. The criteria of this algorithm are "Replace a page that will not be used for the longest period".

The LRU (least recently used) policy replaces the page that has not been used for the longest time in the past and is somewhat of an approximation to the optimal policy. It doesn't suffer from Belady's anomaly (the ordering of the least recently used pages won't change based on how much memory you have) and is one of the more popular policies. To implement LRU, one must maintain the time of access of each page, whichis an expensive operation if done in software by the kernel. Another way to implement LRU is to store the pages in a stack, move a page to the top of the stack when accessed, and evict it from the bottom of the stack. However, this solution also incurs a lot of overhead (changing stack pointers) for every memory access.

Pre-Requisites:

- **Basic idea on Segmentation.**
- **Accessing of memory with paging.**

- **Page replacement techniques.**
- **Virtual Memory techniques.**

Pre-Lab:

Page Replacement Techniques	**FUNCTIONALITY**
FIFO	Replaces the oldest page in memory, simple strategy.
LRU	Replaces the least recently used page for efficient access.
OPTIMAL	Replaces page not needed for the longest time.

In - Lab Task:

1. write a C program to implement the FIFO page replacement algorithm.

```
#include <stdio.h>

#define MAX_FRAMES 3

int main() {
    int pageFrames[MAX_FRAMES];

    int pageReferenceString[] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2};

    int pageFaults = 0;

    int frameIndex = 0;

    for (int i = 0; i < MAX_FRAMES; i++) {
        pageFrames[i] = -1;
    }

    for (int i = 0; i < 12; i++) {
        int currentPage = pageReferenceString[i];
        int pageFound = 0;

        for (int j = 0; j < MAX_FRAMES; j++) {
            if (pageFrames[j] == currentPage) {
                pageFound = 1;
                break;
            }
        }

        if (!pageFound) {
            pageFrames[frameIndex] = currentPage;
            frameIndex = (frameIndex + 1) % MAX_FRAMES;
            pageFaults++;

            printf("Page %d caused a page fault. Page frames: [", currentPage);

            for (int j = 0; j < MAX_FRAMES; j++) {
                printf("%d", pageFrames[j]);
                if (j < MAX_FRAMES - 1) {
                    printf(", ");
                }
            }
            printf("]\n");
        }
    }

    printf("Total page faults: %d\n", pageFaults);

    return 0;
}
```

2. write a C program to implement the LRU page replacement algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_FRAMES 3

int main() {
    int pageFrames[MAX_FRAMES];
    int pageReferenceString[] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2};
    int pageFaults = 0;
    int frameIndex = 0;

    for (int i = 0; i < MAX_FRAMES; i++) {
        pageFrames[i] = -1;
    }

    for (int i = 0; i < 12; i++) {
        int currentPage = pageReferenceString[i];
        int pageFound = 0;

        for (int j = 0; j < MAX_FRAMES; j++) {
            if (pageFrames[j] == currentPage) {
                pageFound = 1;
                for (int k = j; k > 0; k--) {
                    pageFrames[k] = pageFrames[k - 1];
                }
                pageFrames[0] = currentPage;
                break;
            }
        }

        if (!pageFound) {
            if (frameIndex == MAX_FRAMES) {
                pageFrames[MAX_FRAMES - 1] = currentPage;
            } else {
                pageFrames[frameIndex] = currentPage;
                frameIndex++;
            }
            pageFaults++;
        printf("Page %d caused a page fault. Page frames: [", currentPage);

        for (int j = 0; j < MAX_FRAMES; j++) {
            printf("%d", pageFrames[j]);
            if (j < MAX_FRAMES - 1) {
                printf(", ");
            }
        }
        printf("]\n");
    }
  }

  printf("Total page faults: %d\n", pageFaults);

  return 0;
}
```

Post Lab:

1. Write a C program to simulate Optimal page replacement algorithms.

```
#include <stdio.h>
#define MAX_FRAMES 3

int main() {
    int pageFrames[MAX_FRAMES];
    int pageReferenceString[] = {1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7};
    int pageFaults = 0;

    for (int i = 0; i < MAX_FRAMES; i++) {
        pageFrames[i] = -1;
    }

    for (int i = 0; i < 13; i++) {
        int currentPage = pageReferenceString[i];
        int pageFound = 0;

        for (int j = 0; j < MAX_FRAMES; j++) {
            if (pageFrames[j] == currentPage) {
                pageFound = 1;
                break;
            }
        }

        if (!pageFound) {
            int optimalPageIndex = -1;
            int farthestDistance = -1;

            for (int j = 0; j < MAX_FRAMES; j++) {
                int nextPageIndex = i + 1;
                while (nextPageIndex < 13) {
                    if (pageReferenceString[nextPageIndex] == pageFrames[j]) {
                        if (nextPageIndex > farthestDistance) {
                            farthestDistance = nextPageIndex;
                            optimalPageIndex = j;
                        }
                        break;
                    }
```

```
          nextPageIndex++;
            }
          }

          pageFrames[optimalPageIndex] = currentPage;
          pageFaults++;

          printf("Page %d caused a page fault. Page frames: [", currentPage);

          for (int j = 0; j < MAX_FRAMES; j++) {
            printf("%d", pageFrames[j]);
            if (j < MAX_FRAMES - 1) {
              printf(", ");
            }
          }
          printf("]\n");
        }
      }

      printf("Total page faults: %d\n", pageFaults);

      return 0;
    }
```

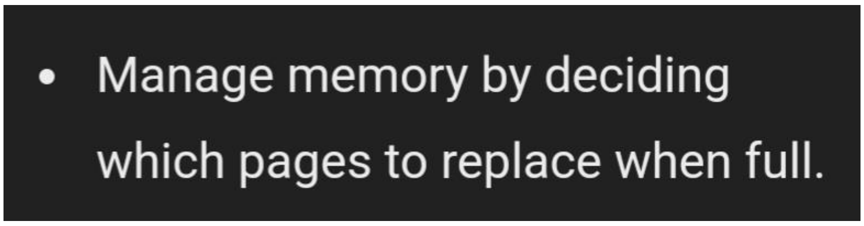
2. Write a C program to simulate LFU page replacement algorithms.

```
#include <stdio.h>
#define MAX_FRAMES 3

typedef struct {
   int page;
   int frequency;
} PageFrame;

int main() {
   PageFrame pageFrames[MAX_FRAMES];
   int pageReferenceString[] = {1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7};
   int pageFaults = 0;

   for (int i = 0; i < MAX_FRAMES; i++) {
      pageFrames[i].page = -1;
      pageFrames[i].frequency = 0;
   }

   for (int i = 0; i < 13; i++) {
      int currentPage = pageReferenceString[i];
      int pageFound = 0;

      for (int j = 0; j < MAX_FRAMES; j++) {
         if (pageFrames[j].page == currentPage) {
            pageFound = 1;
            pageFrames[j].frequency++;
            break;
         }
      }

      if (!pageFound) {
         int minFrequencyIndex = 0;

         for (int j = 1; j < MAX_FRAMES; j++) {
            if (pageFrames[j].frequency < pageFrames[minFrequencyIndex].frequency) {
               minFrequencyIndex = j;
            }
         }
```

```
            pageFrames[minFrequencyIndex].page = currentPage;
            pageFrames[minFrequencyIndex].frequency = 1;
            pageFaults++;

            printf("Page %d caused a page fault. Page frames: [", currentPage);

            for (int j = 0; j < MAX_FRAMES; j++) {
                printf("%d", pageFrames[j].page);
                if (j < MAX_FRAMES - 1) {
                    printf(", ");
                }
            }
            printf("]\n");
        }
    }

    printf("Total page faults: %d\n", pageFaults);

    return 0;
}
```

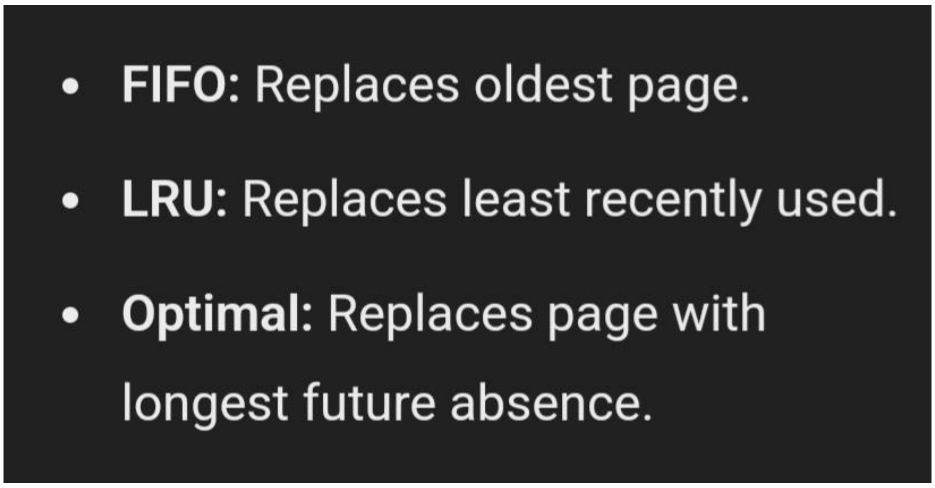
Sample VIVA-VOCE Questions (In-Lab):

1. What is the use of Page Replacement Algorithms?



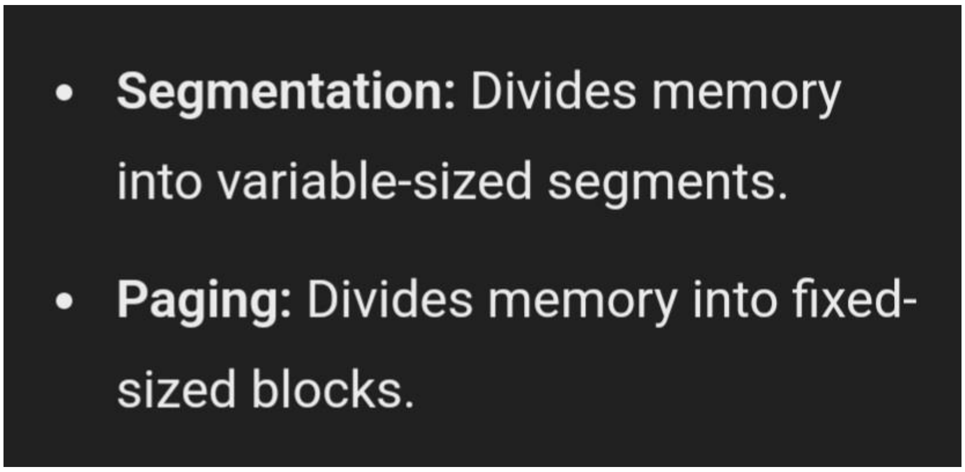
- Manage memory by deciding which pages to replace when full.

2. Differentiate between FIFO, LRU, and Optimal Page Replacement Strategies.



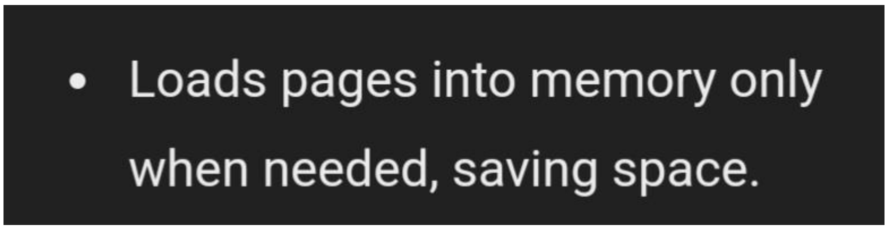
- **FIFO:** Replaces oldest page.
- **LRU:** Replaces least recently used.
- **Optimal:** Replaces page with longest future absence.

3. Explain in detail Segmentation and Paging.



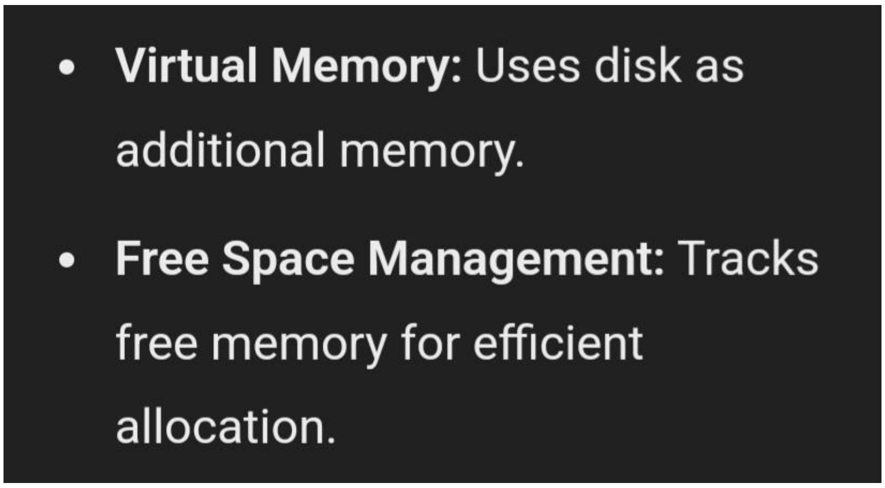
- **Segmentation:** Divides memory into variable-sized segments.
- **Paging:** Divides memory into fixed-sized blocks.

4. Explain in detail Demand Paging.



- Loads pages into memory only when needed, saving space.

5. Explain in detail Virtual Memory and Free Space Management.



- **Virtual Memory:** Uses disk as additional memory.
- **Free Space Management:** Tracks free memory for efficient allocation.

Evaluator Remark (if any):	**Marks Secured_____ out of 50**
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.