1)MULTI THREADING

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUM_THREADS 2

pthread_mutex_t lock;

void *read_input(void *arg) {
    char buffer[256];
    while (1) {
        pthread_mutex_lock(&lock); // Lock the mutex
        printf("Enter some text: ");
        fgets(buffer, sizeof(buffer), stdin);
        printf("You entered: %s\n", buffer);
        pthread_mutex_unlock(&lock); // Unlock the mutex
        sleep(1); // Sleep for a bit before next input
    }
    return NULL;
}

void *print_output(void *arg) {
    while (1) {
        pthread_mutex_lock(&lock); // Lock the mutex
        printf("Output thread is running...\n");
        pthread_mutex_unlock(&lock); // Unlock the mutex
        sleep(2); // sleep for 2 seconds
    }
```

```
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    pthread_mutex_init(&lock, NULL);

    if (pthread_create(&threads[0], NULL, read_input, NULL)) {
        fprintf(stderr, "Error creating input thread\n");
        return 1;
    }

    if (pthread_create(&threads[1], NULL, print_output, NULL)) {
        fprintf(stderr, "Error creating output thread\n");
        return 1;
    }

    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&lock);
    return 0;
}
```

Output

```
Output thread is running...
Output thread is running...

Enter some text: Hello, Copilot!
You entered: Hello, Copilot!
Output thread is running...
```

Output thread is running...

2) FIFO PAGING

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_REFERENCES 100

int main() {
    int frames[MAX_FRAMES], referenceString[MAX_REFERENCES];
    int frameCount, referenceCount, pageFaults = 0;
    int i, j, k, flag, currentIndex = 0;

    printf("Enter the number of frames: ");
    scanf("%d", &frameCount);

    printf("Enter the number of page references: ");
    scanf("%d", &referenceCount);

    printf("Enter the page reference string: ");
    for (i = 0; i < referenceCount; i++) {
        scanf("%d", &referenceString[i]);
    }

    // Initialize frames with -1 (indicating they are empty)
    for (i = 0; i < frameCount; i++) {
```

```
    frames[i] = -1;
}

// FIFO Paging algorithm
for (i = 0; i < referenceCount; i++) {
    flag = 0;
    for (j = 0; j < frameCount; j++) {
        if (frames[j] == referenceString[i]) {
            flag = 1;  // Page found in frame
            break;
        }
    }

    if (flag == 0) {  // Page fault occurs
        frames[currentIndex] = referenceString[i];
        currentIndex = (currentIndex + 1) % frameCount;
        pageFaults++;
    }

    // Print current state of frames
    printf("Frames: ");
    for (k = 0; k < frameCount; k++) {
        if (frames[k] != -1)
            printf("%d ", frames[k]);
        else
            printf("_ ");
```

```
        }
        printf("\n");
    }

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;
}
```

Input

Enter the number of frames: 3

Enter the number of page references: 8

Enter the page reference string: 1 3 0 3 5 6 3 1

Output

Frames: 1 _ _

Frames: 1 3 _

Frames: 1 3 0

Frames: 1 3 0

Frames: 5 3 0

Frames: 5 6 0

Frames: 5 6 3

Frames: 1 6 3

Total Page Faults: 7

3) LRU PAGING

```
#include <stdio.h>

#define MAX_FRAMES 10
```

```
#define MAX_REFERENCES 100

int main() {
    int frames[MAX_FRAMES], referenceString[MAX_REFERENCES];
    int frameCount, referenceCount, pageFaults = 0;
    int i, j, k, flag, leastRecentlyUsed, time[MAX_FRAMES], counter = 0;

    printf("Enter the number of frames: ");
    scanf("%d", &frameCount);

    printf("Enter the number of page references: ");
    scanf("%d", &referenceCount);

    printf("Enter the page reference string: ");
    for (i = 0; i < referenceCount; i++) {
        scanf("%d", &referenceString[i]);
    }

    // Initialize frames and time arrays
    for (i = 0; i < frameCount; i++) {
        frames[i] = -1;
        time[i] = 0;
    }

    // LRU Paging algorithm
    for (i = 0; i < referenceCount; i++) {
        flag = 0;
        for (j = 0; j < frameCount; j++) {
            if (frames[j] == referenceString[i]) {
                flag = 1;  // Page found in frame
                time[j] = ++counter;  // Update time
```

```
            break;
        }
    }

    if (flag == 0) {  // Page fault occurs
        leastRecentlyUsed = 0;
        for (j = 1; j < frameCount; j++) {
            if (time[j] < time[leastRecentlyUsed]) {
                leastRecentlyUsed = j;
            }
        }
        frames[leastRecentlyUsed] = referenceString[i];
        time[leastRecentlyUsed] = ++counter;
        pageFaults++;
    }

    // Print current state of frames
    printf("Frames: ");
    for (k = 0; k < frameCount; k++) {
        if (frames[k] != -1)
            printf("%d ", frames[k]);
        else
            printf("_ ");
    }
    printf("\n");
  }

  printf("Total Page Faults: %d\n", pageFaults);

  return 0;
}
```

INPUT

Enter the number of frames: 3

Enter the number of page references: 8

Enter the page reference string: 1 3 0 3 5 6 3 1

OUTPUT

Frames: 1 _ _

Frames: 1 3 _

Frames: 1 3 0

Frames: 1 3 0

Frames: 1 5 0

Frames: 6 5 0

Frames: 6 3 0

Frames: 6 3 1

Total Page Faults: 6

4)OPTIMAL PAGING

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_REFERENCES 100

int main() {
    int frames[MAX_FRAMES], referenceString[MAX_REFERENCES];
    int frameCount, referenceCount, pageFaults = 0;
    int i, j, k, flag, farthest, future[MAX_FRAMES];

    printf("Enter the number of frames: ");
    scanf("%d", &frameCount);
```

```
printf("Enter the number of page references: ");
scanf("%d", &referenceCount);

printf("Enter the page reference string: ");
for (i = 0; i < referenceCount; i++) {
   scanf("%d", &referenceString[i]);
}

// Initialize frames with -1 (indicating they are empty)
for (i = 0; i < frameCount; i++) {
   frames[i] = -1;
}

// Optimal Paging algorithm
for (i = 0; i < referenceCount; i++) {
   flag = 0;
   for (j = 0; j < frameCount; j++) {
      if (frames[j] == referenceString[i]) {
         flag = 1;  // Page found in frame
         break;
      }
   }

   if (flag == 0) {  // Page fault occurs
      int found = 0;
      for (j = 0; j < frameCount; j++) {
         if (frames[j] == -1) {
            frames[j] = referenceString[i];
            found = 1;
            pageFaults++;
            break;
```

```
        }
    }

    if (!found) {
        for (j = 0; j < frameCount; j++) {
            future[j] = -1;
            for (k = i + 1; k < referenceCount; k++) {
                if (frames[j] == referenceString[k]) {
                    future[j] = k;
                    break;
                }
            }
        }

        farthest = 0;
        for (j = 1; j < frameCount; j++) {
            if (future[j] == -1) {
                farthest = j;
                break;
            } else if (future[j] > future[farthest]) {
                farthest = j;
            }
        }
        frames[farthest] = referenceString[i];
        pageFaults++;
    }
}

// Print current state of frames
printf("Frames: ");
for (k = 0; k < frameCount; k++) {
```

```
            if (frames[k] != -1)
                printf("%d ", frames[k]);
            else
                printf("_ ");
        }
        printf("\n");
    }

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;
}
```

Input

Enter the number of frames: 3

Enter the number of page references: 8

Enter the page reference string: 1 3 0 3 5 6 3 1

output

Frames: 1 _ _

Frames: 1 3 _

Frames: 1 3 0

Frames: 1 3 0

Frames: 1 3 5

Frames: 6 3 5

Frames: 6 3 5

Frames: 6 3 1

Total Page Faults: 6

5)Sequential file allocation

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *file;
    char filename[] = "sequential_file.txt";
    char data_to_write[] = "This is an example of sequential file allocation.";
    char buffer[100];

    // Open file for writing
    file = fopen(filename, "w");
    if (file == NULL) {
        printf("Error opening file for writing.\n");
        return 1;
    }

    // Write data to file
    fprintf(file, "%s", data_to_write);
    fclose(file);

    // Open file for reading
    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file for reading.\n");
        return 1;
    }

    // Read data from file
    fgets(buffer, 100, file);
    printf("Data read from file: %s\n", buffer);
```

```
    fclose(file);

    return 0;
}
```

Input

"This is an example of sequential file allocation."

Output

Data read from file: This is an example of sequential file allocation.