Assignment 6: Producer-Consumer Problem using Semaphores

Code:

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

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int in = 0, out = 0;

sem_t empty, full, mutex;

void *producer(void *param) {
    int item;
    while (1) {
        item = rand() % 100;
        sem_wait(&empty);
        sem_wait(&mutex);

        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
```

```
        sem_post(&mutex);
        sem_post(&full);
    }
}

void *consumer(void *param) {
    int item;
    while (1) {
        sem_wait(&full);
        sem_wait(&mutex);

        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        sem_post(&mutex);
        sem_post(&empty);
    }
}

int main() {
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);
```

```
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    return 0;
}
```

Output:

```
Producer produced: 45
Consumer consumed: 45
Producer produced: 23
Consumer consumed: 23
... (continues infinitely)
```

Assignment 7: Banker's Algorithm for Deadlock Avoidance

Code:

```
#include <stdio.h>

int main() {
    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources

    int alloc[5][3] = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};
    int max[5][3] = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
    int avail[3] = {3, 3, 2};

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
```

```
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("Following is the SAFE Sequence:\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);

    return (0);
}
```

Output:

Following is the SAFE Sequence:

P1 -> P3 -> P4 -> P0 -> P2

Assignment 8: Contiguous Memory Allocation Techniques

Code:

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

void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;

    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
    printf("Process No. Process Size Block no.\n");
```

```
    for (int i = 0; i < n; i++) {
        printf(" %d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0;
}
```

Output:

Process No. Process Size Block no.

1	212	4
2	417	5
3	112	3
4	426	Not Allocated

Assignment 9: Disk Scheduling Algorithms (FCFS and SCAN)

Code for FCFS:

```
#include <stdio.h>

void FCFS(int arr[], int head, int size) {
    int seek_count = 0;
    int distance, cur_track;

    for (int i = 0; i < size; i++) {
        cur_track = arr[i];
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }
    printf("Total seek operations: %d\n", seek_count);
}

int main() {
    int arr[] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;
    int size = sizeof(arr) / sizeof(arr[0]);

    FCFS(arr, head, size);
```

```
    return 0;
}
```

Output:

```
Total seek operations: 510
```