Synchronization Algorithms in Operating Systems (Without Threads)

Semaphore - Mutual Exclusion

```
int mutex = 1;

void wait(int *s) {
    while (*s <= 0);
    (*s)--;
}

void signal(int *s) {
    (*s)++;
}

void critical_section(int id) {
    wait(&mutex);
    printf("Process %d is in critical section\n", id);
    signal(&mutex);
}
```

Output:

```
Process 1 is in critical section
Process 2 is in critical section
```

Peterson's Solution (2 Processes)

```
bool flag[2] = {false, false};
int turn;

void peterson(int i) {
    int j = 1 - i;
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);
    printf("Process %d is in critical section\n", i);
    flag[i] = false;
}
```

Output:

```
Process 0 is in critical section
Process 1 is in critical section
```

Dining Philosophers (5 Philosophers)

```
int fork_status[5] = {1, 1, 1, 1, 1};

void dine(int id) {
    int left = id;
    int right = (id + 1) % 5;

    if (fork_status[left] && fork_status[right]) {
        fork_status[left] = fork_status[right] = 0;
        printf("Philosopher %d is eating\n", id);
        fork_status[left] = fork_status[right] = 1;
    } else {
        printf("Philosopher %d is waiting for forks\n", id);
    }
}
```

Output:

```
Philosopher 0 is eating
Philosopher 1 is eating
```

Reader-Writer Problem

```
int readcount = 0, wrt = 1, mutex = 1;

void wait(int *s) { while (*s <= 0); (*s)--; }
void signal(int *s) { (*s)++; }

void reader(int id) {
    wait(&mutex);
    readcount++;
    if (readcount == 1) wait(&wrt);
    signal(&mutex);
    printf("Reader %d is reading\n", id);
    wait(&mutex);
    readcount--;
    if (readcount == 0) signal(&wrt);
    signal(&mutex);
}

void writer(int id) {
    wait(&wrt);
    printf("Writer %d is writing\n", id);
    signal(&wrt);
}
```

Output:

```
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing
```

Producer-Consumer (1 Slot Buffer)

```
int buffer = 0, mutex = 1, empty = 1, full = 0;

void wait(int *s) { while (*s <= 0); (*s)--; }
void signal(int *s) { (*s)++; }

void produce() {
    wait(&empty);
    wait(&mutex);
    buffer = 1;
    printf("Produced an item\n");
    signal(&mutex);
    signal(&full);
}

void consume() {
    wait(&full);
    wait(&mutex);
    buffer = 0;
    printf("Consumed an item\n");
    signal(&mutex);
    signal(&empty);
}
```

```
Output:
Produced an item
Consumed an item
```