Thread Synchronization with Semaphores

1. Introduction

Concurrent programming is a technique in which two or more processes start, run in an interleaved fashion through context switching and complete in an overlapping time period by managing access to shared resources e.g. on a single core of CPU. The goal is to handle races, synchronization and deadlock conditions among threads of a single process.

2. Problem

N *mcounter* threads count independent incoming messages in a system, and another thread *mmonitor* gets the count of threads at time intervals of size t1, and then resets the counter to 0. The *mmonitor* then places this value in a buffer of size b, and a *mmcollector* thread reads the values from the buffer.

3. Solution

3.1. Design

As shown in Figure 3.1.1, the *mcounter* and the *mmonitor* threads should be able to access the shared integer while the *mmonitor* and the *mcollector* threads should be able to access the buffer.

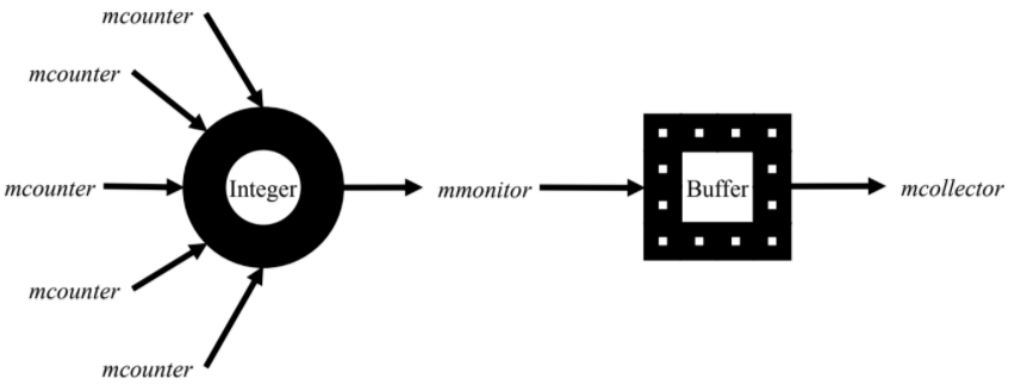


Figure 3.1.1

3.2. Implementation

First of all, this is how the program gets called from the main:

```
int main()
{
    NUMBER_OF_THREADS = 3;
    BUFF_SIZE = 5;
    start();
}
```

I started by initializing a global variable NUMBER_OF_THREADS to hold the number of Counter threads in the process, also initialized a BUFF_SIZE variable to hold the buffer size. A call is then made to `start()` function, and here's how it looks like:

```
void start(){
    init();

    pthread_t mmonitor, mcollector;
    pthread_t mcounter[NUMBER_OF_THREADS];
    if(pthread_create(&mmonitor, NULL, Monitor, NULL) != 0){
        puts("ERROR IN THREADS");
        exit(1);
    }
    if(pthread_create(&mcollector, NULL, Collector, NULL) != 0){
        puts("ERROR IN THREADS");
        exit(1);
    }
    for(int i = 0 ; i < NUMBER_OF_THREADS ; i++){
        if(pthread_create(&mcounter[i], NULL, Counter, i) != 0){
        puts("ERROR IN THREADS");
        exit(1);
        }
    }

    pthread_join(mmonitor, NULL);
    pthread_join(mcollector, NULL);
    for(int i = 0 ; i < NUMBER_OF_THREADS ; i++){
        pthread_join(mcounter[i], NULL);
    }
}
```

A call is made to `init()` function to initialize the main variables in the process, then threads are created for monitor, collector and counter with the number of threads assigned in the main function.

```
void init(){
    counter = 0;
    random_sleep = rand() % 5;
    buffer_in = 0;
    buffer_out = 0;
    inserted = 0;
    t1 = 1;

    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFF_SIZE);
    sem_init(&count, 0, 1);
    sem_init(&b, 0, 1);

    buffer = (int*) malloc(BUFF_SIZE * sizeof(int));
}
```

Counter, Monitor and Collector functions are made to be interleaved and are not allowed to access the shared variables at the same time.

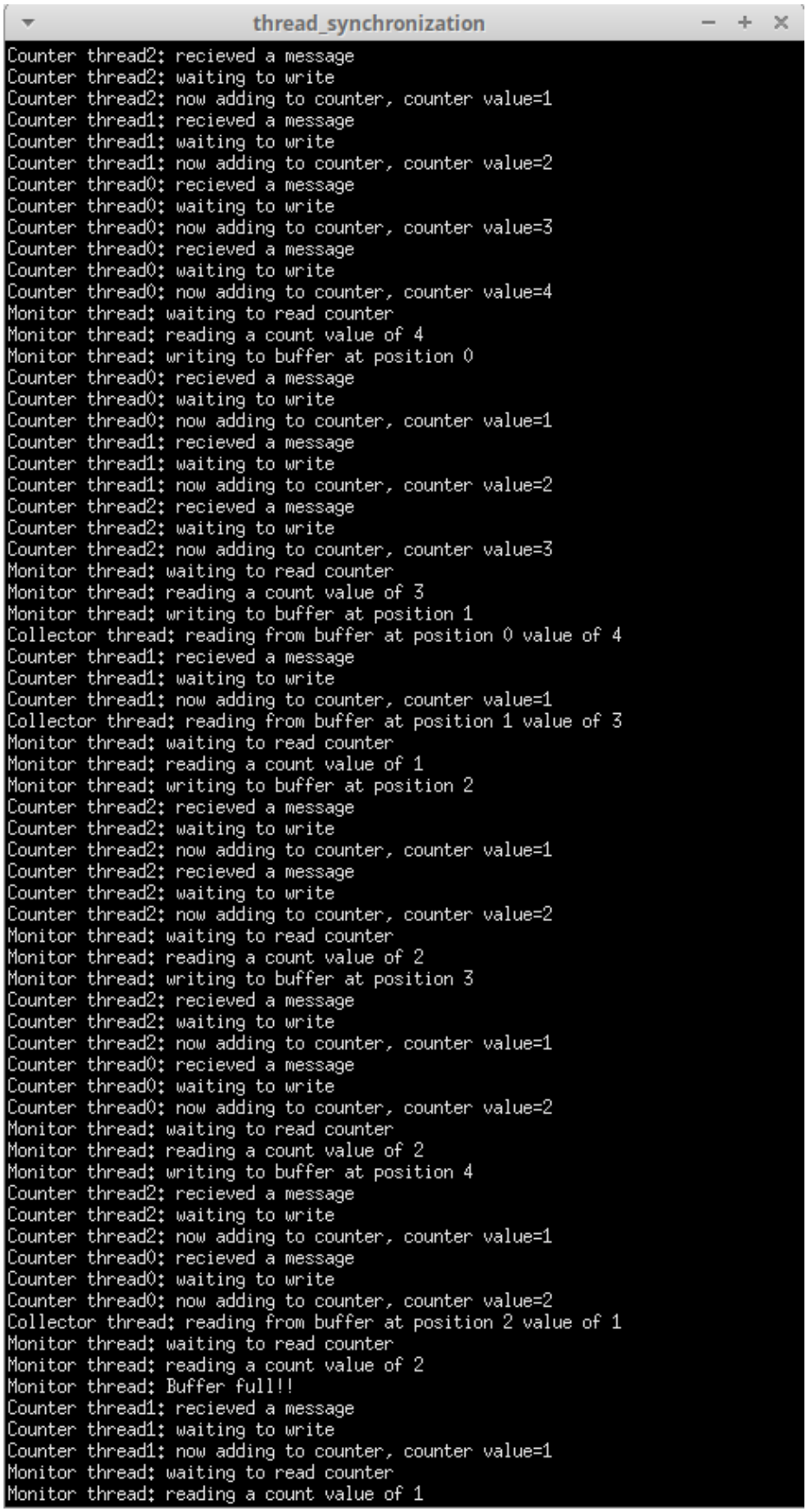
```
void *Counter(void* arg)
{
    int thread = (int) arg;

    while(1){
        random_sleep = rand() % 5;
        printf("Counter thread%d: recieved a message\n", thread);
        sem_wait(&count);
        printf("Counter thread%d: waiting to write\n", thread);
        counter++;
        printf("Counter thread%d: now adding to counter, counter value=%d\n", thread, coun
ter);
        sem_post(&count);
        sleep(random_sleep);
    }
}
void *Monitor(void *arg)
{
    int current_counter;
    while (1)
    {
        random_sleep = rand() % 5;
        sleep(t1);
        printf("Monitor thread: waiting to read counter\n");
        sem_wait(&count);
        current_counter = counter;
        counter = 0;
        printf("Monitor thread: reading a count value of %d\n", current_counter);
        sem_post(&count);

        sem_wait(&empty);
        if (buffer_in == BUFF_SIZE)
        {
            printf("Monitor thread: Buffer full!!\n");
        }
        else
        {
            sem_wait(&b);
            printf("Monitor thread: writing to buffer at position %d\n", buffer_in);
            buffer[buffer_in] = current_counter;
            buffer_in++;
            inserted++;
            sem_post(&b);
        }
        sem_post(&full);
    }
}
```

```
void *Collector(void *arg)
{
    while (buffer_out != BUFF_SIZE)
    {
        random_sleep = rand() % 5;
        sleep(random_sleep);
        if (inserted == 0)
        {
            printf("Collector thread: nothing is in the buffer!\n");
        }
        else
        {
            sem_wait(&b);
            sem_wait(&full);
            printf("Collector thread: reading from buffer at position %d value of %d\n",
buffer_out, buffer[buffer_out]);
            buffer_out++;
            sem_post(&empty);
            sem_post(&b);
        }
    }
    exit(1);
}
```

4. Sample Run


thread_synchronization
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=1
Counter thread1: recieved a message
Counter thread1: waiting to write
Counter thread1: now adding to counter, counter value=2
Counter thread0: recieved a message
Counter thread0: waiting to write
Counter thread0: now adding to counter, counter value=3
Counter thread0: recieved a message
Counter thread0: waiting to write
Counter thread0: now adding to counter, counter value=4
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 4
Monitor thread: writing to buffer at position 0
Counter thread0: recieved a message
Counter thread0: waiting to write
Counter thread0: now adding to counter, counter value=1
Counter thread1: recieved a message
Counter thread1: waiting to write
Counter thread1: now adding to counter, counter value=2
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=3
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 3
Monitor thread: writing to buffer at position 1
Collector thread: reading from buffer at position 0 value of 4
Counter thread1: recieved a message
Counter thread1: waiting to write
Counter thread1: now adding to counter, counter value=1
Collector thread: reading from buffer at position 1 value of 3
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 1
Monitor thread: writing to buffer at position 2
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=1
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=2
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 2
Monitor thread: writing to buffer at position 3
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=1
Counter thread0: recieved a message
Counter thread0: waiting to write
Counter thread0: now adding to counter, counter value=2
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 2
Monitor thread: writing to buffer at position 4
Counter thread2: recieved a message
Counter thread2: waiting to write
Counter thread2: now adding to counter, counter value=1
Counter thread0: recieved a message
Counter thread0: waiting to write
Counter thread0: now adding to counter, counter value=2
Collector thread: reading from buffer at position 2 value of 1
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 2
Monitor thread: Buffer full!!
Counter thread1: recieved a message
Counter thread1: waiting to write
Counter thread1: now adding to counter, counter value=1
Monitor thread: waiting to read counter
Monitor thread: reading a count value of 1