11/9/25

OS

Name: Ayush. Deepak. Chougule.
Rollno: 04 TY CSE D1.
PRN : 1032231720

Assignment - 06

★ Problem Statement:
To implement the readers - Writers problem.

★ Objective
- To understand the concept of process synchronization
- To understand the classical ~~p~~ readers - Writers problem
- To devise a solution using semaphores.

★ Theory:

- Semaphore: A semaphore is an integer value used for process synchronization and mutual exclusion in concurrent systems. semaphores operate through two atomic operations: wait() and signal ().
- Types of semaphore: ① binary
  → values 0 / 1., for mutual exclusion
  ② Counting
  → non-negative resource management
- wait () → decrements, blocks if negative
  signal () → Increments, unblocks if waiting).
- In the readers - writers problem, semaphores ensure multiple ~~data~~ readers can access shared data silumtaneously but only one writer accesses it at a time, preventing data inconsistency and race conditions. This structure helps avoid deadlocks and starvation by managing entry and exit via semaphores.

* Concluision: Thus, we have studied and implemented the co
of process synchronization using ~~seam~~ semap

* ~~Concluisit~~ FAQ's:

1] How can semaphores be used to implement mutual exc
in Accening a critical section.

→ Semaphores are crucial for mutual exclusion in operation
system. By using wait() and signal() operations, accen
a critical section can be restricted so only one process e
at a time, thus preventing race conditions.

2] Discuss producer-consumer problem and devise a solu
using semaphores.

→ In the producer-consumer problem, semaphores control
available resources and empty slots, ensuring produ
and consumers operate in synce without conflict or da
loss.

3]. Describe the role of semaphores in solving the reader-wr
problem.

→ In the readers-writers problem, semaphores allow mul
readers parallel access to shared data but ensure that
writers have exclusive accem when updating data. Th
managed with seprate semaphores for readers and writer
alongside counters tracking active procem, preventing
reader and writer starvation. when a writer is waiting
new reader can start; writers get priority and use
semaphores as a gate to block new readers until the wr

is completed.

4] List and disscuss the different process synchronization mechanisms.

→

- mutexes : lock mechanisms for exclusive access to critical sections allowing one process ~~time~~ at a time

- Semaphores : Counting or binary signals to control access and synchronization between process

- monitors : High level sychronization Construct combining mutexes and condition Variables for control access

- Condition Variables : Used with mutexes to allow processes to wait for certain conditions before proceeding.

- Spin locks : Busy wait locks suited for short waits on shared resources.

- Peterson's Solution : A classical protocol for mutual exclusion between two processes.

★ Input :

Enter initial Value of Shared Variable : 100
Enter number of readers and writers (max 5) : 5

★ Output :

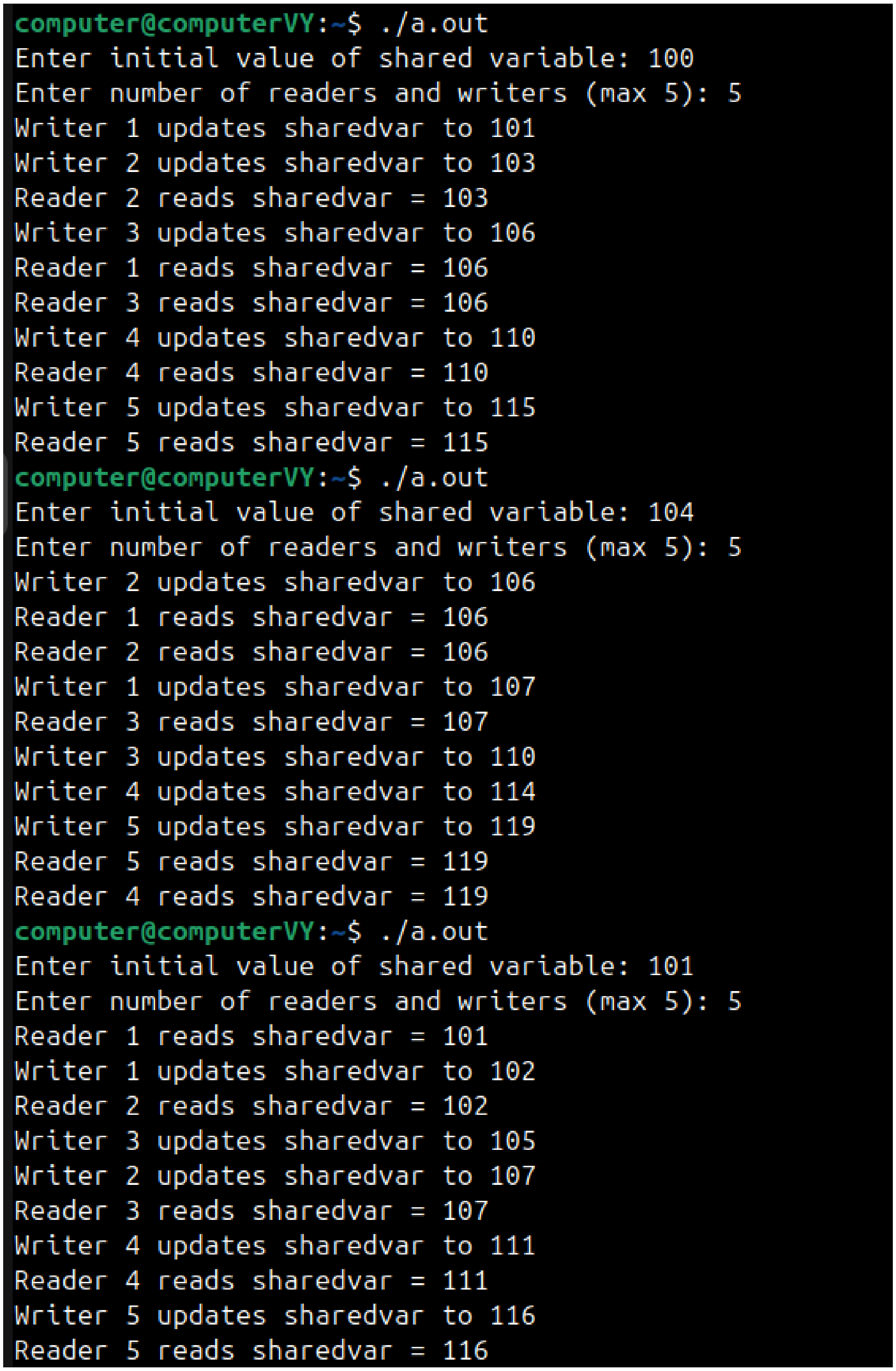
writer 1 ~~outp~~ updates sharedvar to 101
writer 2 updates sharedvar to 103

Reader 2 reads sharedVar = 103
writer 3 · updates sharedVar to 106
Reader 1 ~~write~~ reads sharedVar = 106
Reader 3 reads sharedVar = 106
Writer 4 updates sharedVar to 110
Reader 4 reads sharedVar = 110
writer 5 updates sharedVar to 115
reader 5 reads sharedVar = 115

CODE:

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

sem_t mutex, wrt;
int sharedvar = 99;
int readercount = 0;
pthread_t writers[5], readers[5];

void* reader(void* arg)
{
    int id = (int)(long)arg;

    sem_wait(&mutex);
    readercount++;
    if (readercount == 1)
    {
        sem_wait(&wrt);
    }
    sem_post(&mutex);

    printf("Reader %d reads sharedvar = %d\n", id, sharedvar);

    sem_wait(&mutex);
    readercount--;
    if (readercount == 0)
    {
        sem_post(&wrt);
    }
    sem_post(&mutex);

    return NULL;
}

void* writer(void* arg)
{
    int id = (int)(long)arg;

    sem_wait(&wrt);
    sharedvar += id;
    printf("Writer %d updates sharedvar to %d\n", id, sharedvar);
    sem_post(&wrt);

    return NULL;
}

int main()
{
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);

    printf("Enter initial value of shared variable: ");
    scanf("%d", &sharedvar);

    int n;
    printf("Enter number of readers and writers (max 5): ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        pthread_create(&writers[i], NULL, writer, (void*)(long)(i + 1));
        pthread_create(&readers[i], NULL, reader, (void*)(long)(i + 1));
    }

    for (int i = 0; i < n; i++)
    {
        pthread_join(writers[i], NULL);
        pthread_join(readers[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

OUTPUT:



```
computer@computerVY:~$ ./a.out
Enter initial value of shared variable: 100
Enter number of readers and writers (max 5): 5
Writer 1 updates sharedvar to 101
Writer 2 updates sharedvar to 103
Reader 2 reads sharedvar = 103
Writer 3 updates sharedvar to 106
Reader 1 reads sharedvar = 106
Reader 3 reads sharedvar = 106
Writer 4 updates sharedvar to 110
Reader 4 reads sharedvar = 110
Writer 5 updates sharedvar to 115
Reader 5 reads sharedvar = 115
computer@computerVY:~$ ./a.out
Enter initial value of shared variable: 104
Enter number of readers and writers (max 5): 5
Writer 2 updates sharedvar to 106
Reader 1 reads sharedvar = 106
Reader 2 reads sharedvar = 106
Writer 1 updates sharedvar to 107
Reader 3 reads sharedvar = 107
Writer 3 updates sharedvar to 110
Writer 4 updates sharedvar to 114
Writer 5 updates sharedvar to 119
Reader 5 reads sharedvar = 119
Reader 4 reads sharedvar = 119
computer@computerVY:~$ ./a.out
Enter initial value of shared variable: 101
Enter number of readers and writers (max 5): 5
Reader 1 reads sharedvar = 101
Writer 1 updates sharedvar to 102
Reader 2 reads sharedvar = 102
Writer 3 updates sharedvar to 105
Writer 2 updates sharedvar to 107
Reader 3 reads sharedvar = 107
Writer 4 updates sharedvar to 111
Reader 4 reads sharedvar = 111
Writer 5 updates sharedvar to 116
Reader 5 reads sharedvar = 116
```