Student Name: Ahmet Furkan Kavraz Stundet Number: 150190024

REPORT

In homework, we were asked to find a solution to an ordinary news source-subscriber problem. I use processes for all news-sources and subscribers. Every process is a news-source or a subscriber byself. I use published process-id instead of published data, it is storing at a shared memory space. The shared memory space and the data type can be changeable.

There are two functions one is read_news() and the other one is publish(). read_news() function will called by subscribers and publish() function will called by news sources. I want to examine functions one by one.

Firstly for publish() function, things we are asked to pay attention to:

--> There exist a published data, which is still in process? The new news-source should wait until the previous data is read by each subscriber.

- To solve turqouise problem, we need to use a semaphore for mutual exclusion.
- To solve green problem, we need to use a semaphore for synchronization, the publish() function should wait all subscribers to finish, before giving a finish signal for mutex.

Secondly, for read_news() function, things we are asked to pay attention to:

--> If there is no published data, it will wait.

--> Every subscriber fetch a copy only one time.

- To solve yellow problem, we need to use a semaphore for synchronization, the read_news() function should until the new news-source published.

- To solve green problem, we need to use a shared memory space and a variable. They is used for understanding the processes that reached the news-source before. Since every process has its own variables, the variable will be different for every subscriber.

To summarize in one piece:

```
void read_news(){

    p(read_sem, 1); //waiting the news-source

    if (the_subscriber_did_not_receive_the_published_data_before){

        print_the_received_data;
        the_subscriber_receive_the_published_data;

        v(publisher_sem, 1); //signal for the news-source
    }
    else  {
        v(read_sem, 1);//incrementing because it will already reach

    }
}
```

```
void publish(){

    p(publish_mutex, 1); //mutex between news-source

    publishing_data_to_the_shared_memory_space;
    clearing_the_received_subscriber_shared_memory_space;

    v(read_sem, n); //signal for the subscribers
    p(publisher_sem, n); //waiting to finish all subscribers

    v(publish_mutex, 1); //mutex between news-source
}
```