OS Project1 README 2019/2020

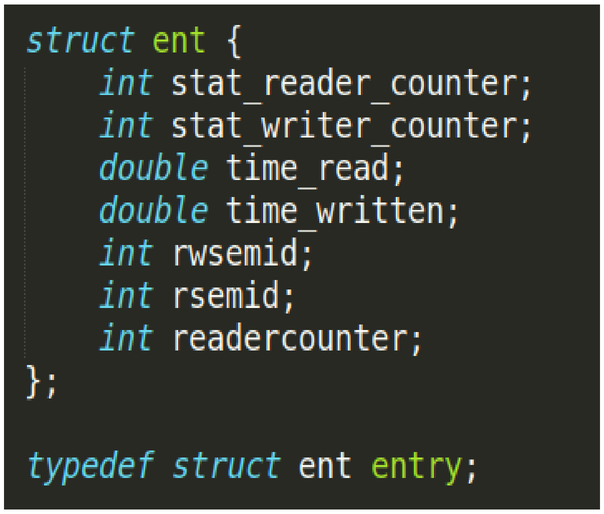
Thodoris Maximilianos Chytis sdi1700197

This project successfully tackles the task of process synchronization between process readers and writers attempting to access a shared memory's contents. The program is given the number of peer processes to be created, the number of entries the shared memory must consist of as well as the ratio of readers/writers.

Files:

- source: coordinator.c sharedmemory.c semaphores.c
- headers: shared memory.h semaphores.h
- makefile: Makefile

The first shared memory between the peer processes contains data of type entry. Each entry consists of counters which track how many times the entry has been read or been written on, counters which track the amount of time the entry has been read or been written on, and last two semaphores and a variable counting the readers in the entry at the exact time to solve the synchronization problem.



```
struct ent {
    int stat_reader_counter;
    int stat_writer_counter;
    double time_read;
    double time_written;
    int rwsemid;
    int rsemid;
    int readercounter;
};

typedef struct ent entry;
```

Another shared memory segment is created in order to have some int variables available which are visible and useful to all the processes while they run at the same time.

The peer child processes are created using the fork system call, copying the code of the parent, into another data segment and running it autonomously after the fork call. Their attempting to access the shared memory will be looped “execution_num” of times which is defined in the coordinator source code.

The choice for a process to be a reader or a writer is handled randomly as well as the entry chosen to attempting to access.

The semaphores and the readercounter of the entry are used to block a writer ,and allow access to a reader if the entry is being read.
The rwsemaphore is used to block a wrter/reader ifthe entry is being written on.

Inside the Critical Section the process sleeps for an amount of time about true to a user’s reading or writing time using an exponential distribution function.

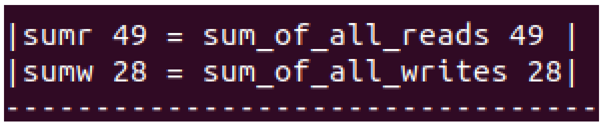
After completing the number of execution loops each child process print out information about the numbers of times it has read, written on an entry ass well as the average waiting time before entering entries being blocked by the semaphore. Then the child process exits successfully.

The father process is waiting for all child processes to terminate,before accessing the shared memory and deleting the semaphores. After that both shared memory processes are being detached and deleted and the father process,the program terminate successfully.

<u>Compilation:</u>

1)//execution_num=7

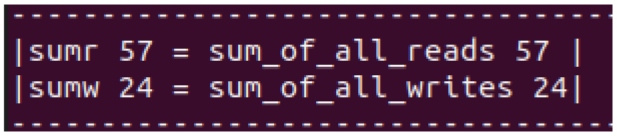
```
make
./ReadersWriters  11 5 0.6
make clean
```



```
|sumr 49 = sum_of_all_reads 49 |
|sumw 28 = sum_of_all_writes 28|
--------------------------------
```

2)//execution_num=3

```
make
./ReadersWriters 27 11 0.7
make clean
```



```
--------------------------------
|sumr 57 = sum_of_all_reads 57 |
|sumw 24 = sum_of_all_writes 24|
--------------------------------
```

As we would have guesses we get two equations at that end indicating that everything is stable because,

- sumr:
  the sum of all the times all processes have read an entry
- sumw:
  the sum of all the times all processes have written on an entry
- sum_of_all_reads:
  the sum of all the times all entries have been read
- sum_of_all_writes:
  the sum of all the times all entries have been written on