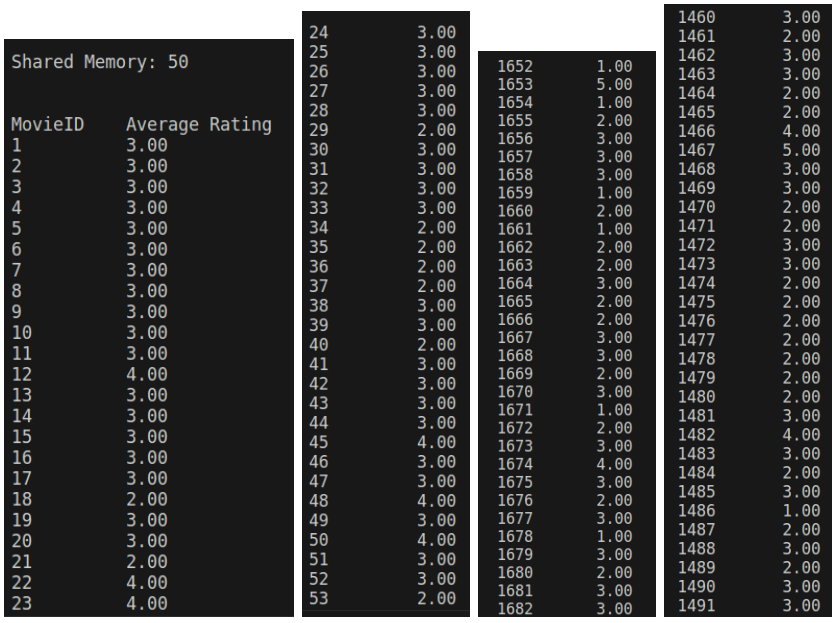
LAB 2: Process

LÔ HOÀNG BẢO - 2252066 - CN01

Ngày 04 tháng 04 năm 2024

Exercise 1

Write a program that spawns two child processes, and each of them will read a file and compute the average ratings of movies in the file and implement the program by using shared memory method.



```
Shared Memory: 50

MovieID    Average Rating
1          3.00
2          3.00
3          3.00
4          3.00
5          3.00
6          3.00
7          3.00
8          3.00
9          3.00
10         3.00
11         3.00
12         4.00
13         3.00
14         3.00
15         3.00
16         3.00
17         3.00
18         2.00
19         3.00
20         3.00
21         2.00
22         4.00
23         4.00
24         3.00
25         3.00
26         3.00
27         3.00
28         3.00
29         2.00
30         3.00
31         3.00
32         3.00
33         3.00
34         2.00
35         2.00
36         2.00
37         2.00
38         3.00
39         3.00
40         2.00
41         3.00
42         3.00
43         3.00
44         3.00
45         4.00
46         3.00
47         3.00
48         4.00
49         3.00
50         4.00
51         3.00
52         3.00
53         2.00
1652       1.00
1653       5.00
1654       1.00
1655       2.00
1656       3.00
1657       3.00
1658       3.00
1659       1.00
1660       2.00
1661       1.00
1662       2.00
1663       2.00
1664       3.00
1665       2.00
1666       2.00
1667       3.00
1668       3.00
1669       2.00
1670       3.00
1671       1.00
1672       2.00
1673       3.00
1674       4.00
1675       3.00
1676       2.00
1677       3.00
1678       1.00
1679       3.00
1680       2.00
1681       3.00
1682       3.00
1460       3.00
1461       2.00
1462       3.00
1463       3.00
1464       2.00
1465       2.00
1466       4.00
1467       5.00
1468       3.00
1469       3.00
1470       2.00
1471       2.00
1472       3.00
1473       3.00
1474       2.00
1475       2.00
1476       2.00
1477       2.00
1478       2.00
1479       2.00
1480       2.00
1481       3.00
1482       4.00
1483       3.00
1484       2.00
1485       3.00
1486       1.00
1487       2.00
1488       3.00
1489       2.00
1490       3.00
1491       3.00
```

The images above is the result after running code of Exercise 1.

In my program, I use function compare_and_swap(int *value, int expected, int new_value) for assign value atomically. And i use *shmtget* function to create shared memory region, *shmat* function to create shared segment for shared_memory_pointer.

My program use a pointer shared_memory_pointer to save the sum of rating and the number of user rate for each movie. The index i from 0 to 2003 is for the movieID, shared_memory_pointer[i] is the sum of rating for movieID i and shared_memory_pointer[i + 2004] is the number of user rate for movieID i.

I create child_a and child_b process by using fork() system call. And then child_a will read a file and compute the average ratings of movies in file movie-100k_1.txt. And child_b will handle file movie-100k_2.txt. Process child_a and process child_b will use the same shared_memory_pointer array to store information.

Exercise 2

Write two programs implementing algorithm describe above: one serial version and one multi-thread version. The program takes the number of threads and n from user then creates multiple threads to calculate the sum. Put all of your code in two files named "sum_ serial.c" and "sum_ multi-thread.c". The number of threads and n are passed to your program as an input parameter. The multi-thread version may improve speed-up compared to the serial version. There are at least 2 targets in the Makefile sum_ serial and sum_ multi-thread to compile the two program.

```
noobcoder123@noobcoder123-virtual-machine:~/Desktop/Lab 2/Lab 2 Submission$ ./sum_serial 1000000
sum(1000000) = 1 + 2 + .... + 1000000 = 500000500000
```

The images above is the result after running sum_ serial.c with $n = 1000000$.

In sum_serial program, I calculate the sum normally by using the for loop.

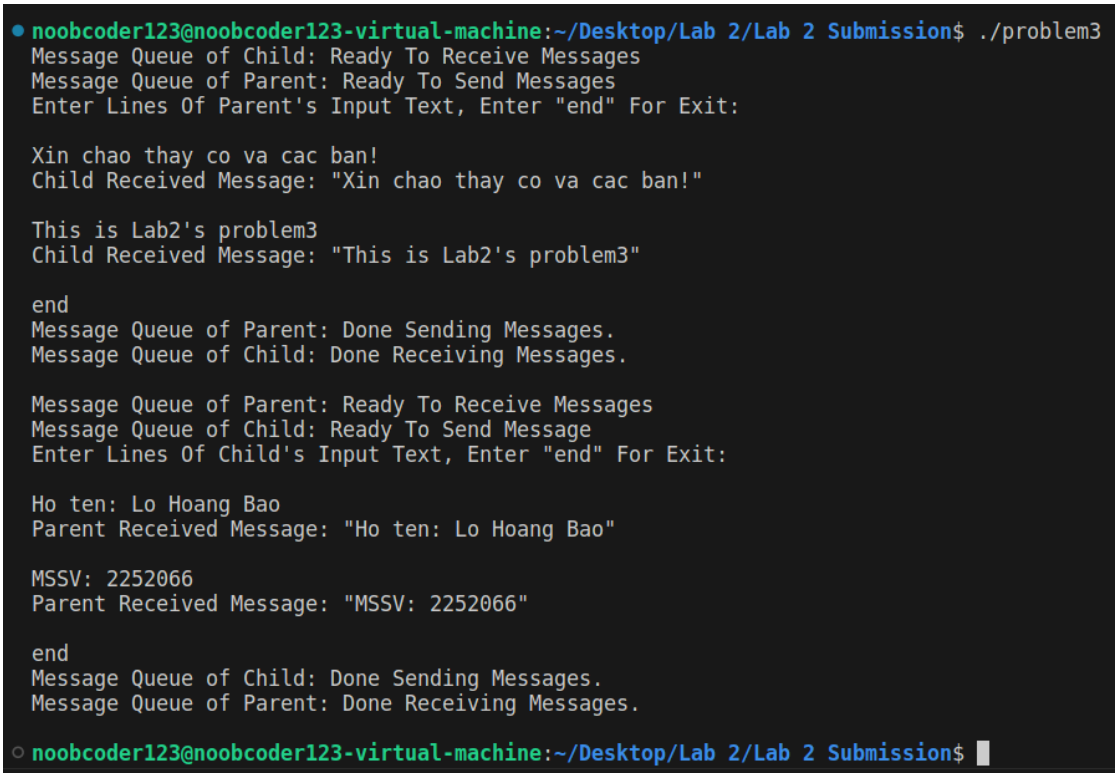
```
noobcoder123@noobcoder123-virtual-machine:~/Desktop/Lab 2/Lab 2 Submission$ ./sum_multi-thread 10 1000000
sum(1000000) = 1 + 2 + .... + 1000000 = 500000500000
```

The images above is the result after running sum_multi-thread.c with $numThreads = 10$ and $n = 1000000$.

In sum_multi-thread program, I use function $runner(void * param)$ to calculate the sum for each thread. And use $thread\_data$ struct to save the range $[upper - jump + 1; upper]$ and $local\_sum$ to save the sum after calculation the sum of segment.

Exercise 3

Implement two-way communication by using multi-thread mechanism



The images above is the result after running code of Exercise 3.

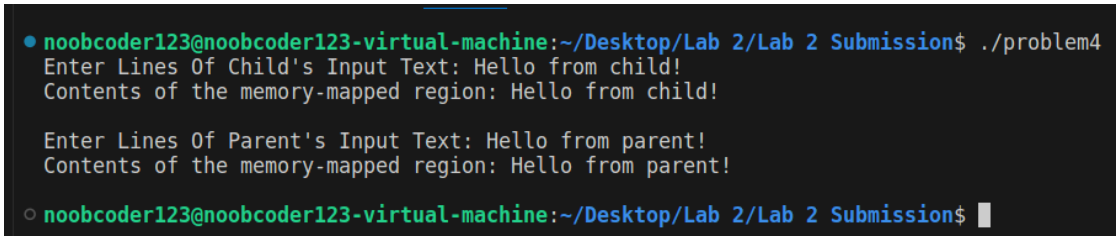
In this exercise, I create 4 functions WRITE_TO_PARENT, READ_FROM_PARENT, WRITE_TO_CHILD, READ_FROM_CHILD.

- WRITE_TO_PARENT: child process send parent process a message through message queue
- READ_FROM_PARENT: child process read message sent by parent process from message queue
- WRITE_TO_CHILD: parent process send child process a message through message queue
- READ_FROM_CHILD: parent process read the message sent by child process from message queue

In this code, I use PERMISSION is 0644 to get read and write the file permission. And use *msgget* function to get message queue ID. $message\_queue\_ID\_1$ variable will use for execute parent process write and child process read. $message\_queue\_ID\_2$ variable will use for execute parent process read and child process write.

Exercise 4

Use mmap to implement the mapping created file into local address space. After the address range mapping, use it as a demonstration for data sharing between the two processes.



The images above is the result after running code of Exercise 4.

In this code, I use *mmap* function create a memory-mapped region for the file. And use *munmap* to unmap the memory and close the file.

I also create WRITE and READ function.

- WRITE: write the data to memory-mapped region
- READ: read the data in memory-mapped region