CSE 344 – SYSTEM PROGRAMMING

HW4

Threads and System V semaphores

151044072 – ALPER YAŞAR

Purpose of this Project?

Synchronization between processes using system v semaphores. There is one supplier thread and multiple consumer threads. The supplier brings materials, one by one. And the consumers consume them, two by two.

Each actor will be modeled by its own thread .Program calling and arguments

Firstly reading arguments. In program there must 7 arguments. There are 2 semaphore. Checking arguments is okey. N must be bigger than 1 and C must be bigger than 4. If this not okey then giving error and exiting.

Then generating a key for creating semaphore. Key are loading in makefile. And creating semaphores and mutual exclusions.

After the semaphores creation creating threads as giving number In C. Each threads will wait until reading. Supplier thread reading file. If file does not contain NxC character “1” and “2” giving error and exiting.

For every character/material read from the file, it will augment/post a semaphore representing its amount. If it reads a ‘1’ it will post the semaphore representing the amount of ‘1’s read so far, and if it reads a ‘2’ it will post the semaphore representing the amount of ‘2’s read so far. It will terminate once it reaches the end of the file.

Each consumer will have its own thread (not detached). Its task is to loop N times. At each iteration it will take/remove one ‘1’ and one ‘2’ by reducing the corresponding semaphores’ values. If either is not available (e.g. if there is no ‘1’ or ‘2’) then it will not take the other. In other words, it will either take two items (one ‘1’ and one ‘2’) or wait until two (one ‘1’ and one ‘2’) are available.

Each consumer thread waiting each other but supplier does not waiting. Values incrementing/changing by semctl.

If values are less than 1 then not release mutex.

```
Supplier: read from input a ‘1’. Current amounts: 4 x ‘1’, 5 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 5 x ‘1’, 5 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 5 x ‘1’, 5 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 6 x ‘1’, 5 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 6 x ‘1’, 5 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 6 x ‘1’, 6 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 6 x ‘1’, 6 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 7 x ‘1’, 6 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 7 x ‘1’, 6 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 8 x ‘1’, 6 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 8 x ‘1’, 6 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 8 x ‘1’, 7 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 8 x ‘1’, 7 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 8 x ‘1’, 8 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 8 x ‘1’, 8 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 9 x ‘1’, 8 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 9 x ‘1’, 8 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 9 x ‘1’, 9 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 9 x ‘1’, 9 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 9 x ‘1’, 10 x ‘2’.
Supplier: read from input a ‘1’. Current amounts: 9 x ‘1’, 10 x ‘2’.
Supplier: delivered a ‘1’. Post-delivery amounts: 10 x ‘1’, 10 x ‘2’.
The Supplier has left.
Consumer-2 at iteration 0 (waiting). Current amounts: 5 x ‘1’, 6 x ‘2’.
Consumer-2 at iteration 0 (consumed). Post-consumption amounts: 10 x ‘1’, 10 x ‘2’.
Consumer-2 at iteration 1 (waiting). Current amounts: 10 x ‘1’, 10 x ‘2’.
Consumer-2 at iteration 1 (consumed). Post-consumption amounts: 9 x ‘1’, 9 x ‘2’.
Consumer-2 at iteration 2 (waiting). Current amounts: 9 x ‘1’, 9 x ‘2’.
Consumer-2 at iteration 2 (consumed). Post-consumption amounts: 8 x ‘1’, 8 x ‘2’.
Consumer-3 at iteration 3 (waiting). Current amounts: 8 x ‘1’, 8 x ‘2’.
Consumer-3 at iteration 3 (consumed). Post-consumption amounts: 7 x ‘1’, 7 x ‘2’.
Consumer-3 has left.
Consumer-4 at iteration 1 (waiting). Current amounts: 7 x ‘1’, 7 x ‘2’.
Consumer-4 at iteration 1 (consumed). Post-consumption amounts: 6 x ‘1’, 6 x ‘2’.
Consumer-1 at iteration 0 (consumed). Post-consumption amounts: 4 x ‘1’, 3 x ‘2’.
Consumer-4 at iteration 2 (waiting). Current amounts: 5 x ‘1’, 5 x ‘2’.
Consumer-4 at iteration 2 (consumed). Post-consumption amounts: 4 x ‘1’, 4 x ‘2’.
Consumer-2 at iteration 3 (waiting). Current amounts: 6 x ‘1’, 6 x ‘2’.
Consumer-2 at iteration 3 (consumed). Post-consumption amounts: 3 x ‘1’, 3 x ‘2’.
Consumer-2 has left.
Consumer-1 at iteration 1 (waiting). Current amounts: 4 x ‘1’, 4 x ‘2’.
Consumer-1 at iteration 1 (consumed). Post-consumption amounts: 3 x ‘1’, 3 x ‘2’.
Consumer-1 at iteration 2 (waiting). Current amounts: 3 x ‘1’, 3 x ‘2’.
Consumer-1 at iteration 2 (consumed). Post-consumption amounts: 2 x ‘1’, 2 x ‘2’.
Consumer-1 at iteration 3 (waiting). Current amounts: 2 x ‘1’, 2 x ‘2’.
Consumer-1 at iteration 3 (consumed). Post-consumption amounts: 1 x ‘1’, 1 x ‘2’.
Consumer-1 has left.
Consumer-4 at iteration 3 (waiting). Current amounts: 1 x ‘1’, 1 x ‘2’.
Consumer-4 at iteration 3 (consumed). Post-consumption amounts: 0 x ‘1’, 0 x ‘2’.
Consumer-4 has left.
viper@ubuntu:~/Desktop/System programming/2022/hw4/151044072$
```