CSE 344
SYSTEMS PROGRAMMING
SPRING 2021

MIDTERM HOMEWORK REPORT

TURKER TERCAN
171044032

HOMEWORK'S CHALLANGE:

- Trying to solve complex producer and consumer problem with maximum 7 semaphores.
- Free to use anything we have learnt up to now.
- We're going to emulate a covid –19 vaccination processes.
- Our input file will contain enough vaccines for both first and second vaccine shots.
- Nurses will read the input file by 1 character at a time, and it'll write it to a shared buffer.
- Vaccinators will consume the buffer by taking a first vaccine and a second vaccine at a single time and then, it will communicate with oldest registered citizen.
- If a citizen gets both 1^{st} and 2^{nd} shots t times, citizen terminates.
- If all the citizens terminate, program finishes successfully.

DESIGN CHOICES:

Communication and Synchronization:

- 7 different semaphores
  - Empty Semaphore: To check if the buffer size is full. If it is full don't let any Nurses to write to buffer.
  - Full Semaphore: To check if the buffer size is empty. If it is empty don't let any Vaccinators to consume any item from the buffer.
  - First Vaccine Semaphore: It contains number of 1^{st} vaccines in the buffer.If buffer is not containing any 1^{st} vaccine, don't let any Vaccinator to continue. It will need both 1^{st} and 2^{nd}.
  - Second Vaccine Semaphore: It works like first vaccine semaphore. Only difference is it contains number of 2^{nd} vaccines.
  - Mutex – 1: In order to protect critical region shared between Nurses and Vaccinators. Only one of them can consume or produce item at a time.
  - Mutex – 2: I used shared memory segments to store citizen's pid, how many shots it needs and if it is receiving a shot at the moment. So, to protect this shared memory segment between Vaccinators and don't letting them to vaccinate same person, I used a mutex.
  - Ready to go Semaphore: Each Nurse, Vaccinator and Citizen processes starts differently. If the Citizen size is large, there is chance that a citizen may finish before all the Vaccinators start. This condition is a race condition at causes a deadlock because I created for each Citizen a fifo to communicate between the Vaccinators and itself. So, if it finishes before every other Vaccinator opens its fifo, Fifo would not have any reading end. So, every Vaccinator would be blocked forever. Each Nurse, Vaccinator and Citizen process, after necessary implementation like open up to shared memory etc., it will write to Clinics fifo. After all the writing is done, clinic will raise this semaphore.

- SIGINT and SIGUSR1 signals
  - SIGINT: I saved all the Nurse, Vaccinator and Citizen's pids in the arrays. If a CTRL + C signal (SIGINT) arrives, Client sends SIGINT signal to all the child processes it has, then, they terminate immediately.
  - SIGUSR1: After all the Citizens have received their shots, all the processes should have finished. So, if all Citizens terminate, Clinic sends SIGUSR1 signal to Nurses and Vaccinators which means that they should terminate immediately.

- FIFO
  - I created a fifo for each Citizen to communicate with between Vaccinators.
    A Vaccinator sends message that it contains how many Vaccine-1 and Vaccine-2 that buffer has and the how many Citizens are remaining to getting shots.
  - And I created a fifo for the Clinic so that every child can message to Clinic to state that it is ready.

Shared Memory Segments:

- Shared File Descriptor: I used fork and exec for creating and running different types of processes. After exec, each Nurse must share a common file descriptor for reading from the input file.

- Shared Buffer: I used an integer array that is size is specified in the arguments. Nurses and Vaccinators must share it for the consuming and producing vaccines.

- Shared Buffer Size: While Nurses are producing and Vaccinators consuming, they should have known the shared buffer's size.

- Citizen Array: Vaccinators must know the which citizen is oldest, how many shots it is still waiting for, and it is receving a shot now (If it is busy).

Code for this type struct:

```
struct reg_citizen {
        pid_t pid;
        int count;
        int busy_flag;
}
```

When the Clinic is creating Citizen processes, It will store each Citizen's pid in the array with the increasing order. So that, the oldest process (small pid) should have been get shot first.

- Vaccine Types: I indicated that I used for semaphores for how many 1st and 2nd vaccine we had but, since it's a semaphore's value is not always correct. Since Vaccinators may hold the semaphore value. So. I created shared memory for storing how many 1st and 2nd vaccines we had and printing them correctly.

clinic.c

- Program starts with the clinic. It handles and manages resources, initializations of Nurses, Citizens, Vaccinators, initializes semaphores, fifos and shared memory segments.

- First it checks the arguments are valid or not by looking at below conditions.

  n >= 2: the number of nurses (integer)

  v >= 2: the number of vaccinators (integer)

  c >= 3: the number of citizens (integer)

  b >= tc+1: size of the buffer (integer),

  t >= 1: how many times each citizen must receive the 2 shots (integer),

  i: pathname of the input file

- It creates a fifo for each Citizen with this template.
  - /tmp/process_pid

- And then, it forks and execs for Nurse, Citizen and Vaccinator processes. For the further use of the child processes, it sends necessary arguments for them.

- Then it waits for all the child processes to terminate or a SIGINT signal. If SIGINT arrives, it sends SIGINT signal to all children processes. If all the Citizen processes have been terminated, It sends SIGUSR1 signal for Nurse and Vaccinators and then, they will be terminated too. And after that it closes all the managed resources, semaphores, etc. And the program ends.

nurse.c

- Nurse is very similar to classic producer. Only difference is the is increments the semaphore for the First and Second Vaccine Semaphore according to which vaccine it produces.
- And it has a a signal handler for both SIGINT and SIGUSR1.

vaccinator.c

- I divided the code to two parts. First part is the consume from the buffer both 1^{st} and 2^{nd} vaccine and I used first semaphore for this critical region.
- Second part is selecting oldest person in the shared citizen array. And it sends a message via chosen's FIFO and makes the boolean value for the chosen person 1 and that means that it is busy right now. While the chosen is busy, no other vaccinators cannot choose it. After the message is sent, vaccinator sets 0 to busy part and decrements the count of how many vaccines the chosen citizen gets by 1.

citizen.c

- It waits for the message from its fifo.
- After message arrives, it prints the message that it got shot for the which time and how many vaccines the buffer has.
- If it got all the shots that is needed, it terminates.

TESTS AND VALGRIND:

I tried some many tests for this program. Since the outputs of the program is large, I saved this test cases in the below .txt files

Test 1:

> log_4c_2t.txt

```
ttwicer@ttwicer-tuf:~/CLionProjects/SystemProgramming/Midterm$ make
gcc nurse.c -o nurse -Wall -ggdb3 -lrt -lpthread -pipe
gcc vaccinator.c -o vaccinator -Wall -ggdb3 -lrt -lpthread -pipe
gcc citizen.c -o citizen -Wall -ggdb3 -lrt -lpthread -pipe
gcc clinic.c -o program -Wall -ggdb3 -lrt -lpthread -pipe
ttwicer@ttwicer-tuf:~/CLionProjects/SystemProgramming/Midterm$ ./program -n 2 -v 2 -c 4 -b 9 -t 2 -i test.txt > log_4c_2t.txt
```

Test 2:

> valgrind_log_4c_2t.txt

```
ttwicer@ttwicer-tuf:~/CLionProjects/SystemProgramming/Midterm$ valgrind -s --leak-check=full --track-origins=yes
 --show-leak-kinds=all --trace-children=yes --log-file=valgrind_log_4c_2t.txt ./program -n 2 -v 2 -c 4 -b 9 -t 2
 -i test.txt
```

Test 3:

> log_8c_4t.txt

```
./program -n 4 -v 4 -c 8 -b 33 -t 4 -i test.txt > log_8c_4t.txt
```

Test 4:

> log_32c_4t.txt

```
./program -n 6 -v 6 -c 32 -b 129 -t 4 -i test.txt > log_32c_4t.txt
```

Test 5:

> log_64c_8t.txt

```
./program -n 10 -v 10 -c 64 -b 513 -t 8 -i test.txt > log_64c_8t.txt
```

Test 6:

> log_256c_8t.txt

```
./program -n 20 -v 20 -c 256 -b 2049 -t 8 -i test.txt > log_256c_8t.txt
```

Test 7:

> log_512c_64t.txt

```
./program -n 20 -v 20 -c 512 -b 32769 -t 64 -i test.txt > log_512c_64t.txt
```

Final Notes:

- I tried some many times for these tests and none of them is failed with a input that has $2 * t * c$ vaccines. And I also tried it above tests some many times with my bash script it worked perfect too. But if a deadlock happens. Please kill the program with CTRL + C signal and run again.
- I completed the optional bonus part.
- I think I adhered to every constraint you stated in the assignment and every attribute you specified.
- The only problem that I couldn't solve was if the number of the citizens is larger than 1024, the citizens that is created after that 1024^{th} citizen, their fifos doesn't work. I guess the limit how many fifos a program create is 1024.