System MIDTERM

Sezer Demir

May 2, 2021

1 Requirements

For this homework I achieved all the requirements those are mentioned in homework includes additional bonus part.

2 Design Decisions

a.) I used fifo structure for communication between vaccinators and citizens. For each citizen program creates a fifo in program's current path. Via those fifo vaccinators send their pid numbers and current clinic vaccine supplies information.

b.) I used 6 shared memory. 1 of them for clinic. I store vaccines in it. Second of them is for fifo names. Vaccinators access citizens fifo names with that memory. Third one is for storing the vaccinator total vaccine dose information. At the end parent process just checks that memory and prints all vaccinator's dose information. Forth one is to store number of total alive citizens. Fifth one to store file current location. Nurses know where last nurse leaved the file via that memory. And last one to store process id of a special cleaner process which I'll mention in 'HOW IT WORKS' section of the report.

c.) I must have used 8 semaphores. 1 of them to control underflow situation, 1 of them to control overflow situation, 1 of them to control clinic buffer shared memory, 1 of them to control fifonames shared memory. 1 of them to control vaccinator shared memory .
In the program parent process fills the fifonames shared memory with fifonames of citizens. So all other processes must have wait the parent process for that. Otherwise before parent process fills it, other processes might want to change it or access it and that would cause problems. So I used another one for that.
One of them for vaccinator control. Basically in my solution if there is no available fifo at the moment, vaccinator must wait until one

of them gets available. Because more than 1 vaccinator cannot invite same citizen at the same time that causes fifo message issues, sending more than 1 message etc., so I used 1 for that too. And at the end last one to control citizen shared memory which is stores number of alive citizens.

d.) I didn't check the content of the file since it is guaranteed that input file contains exactly c*t times 1 and 2. Since I used named semaphores and shared memories, if input file has not proper content then program doesn't work properly and you must remove those semaphores and shared memories manually or restart the OS to run program again.

3 How It Works

3.1 Nurses

Nurses firstly tries to enter clinic buffer. Clinic buffer wrapped up with sem_buf semaphore and sem_overflow semaphore. sem_overflow semaphore initially set to size of clinic buffer so nurses can just carry amount of buffer size vaccines to clinic from the very start. Any nurse process, that want to access, must use those semaphores and decrease their values. And after that nurse in the clinic first checks the current file location by looking at filelocation shared memory. Lets say the nurse before that nurse leaved the file at 15th character, then current nurse gets that number and gets 16th character from file and puts it in clinic buffer, and after that nurse just changes the number in filelocation shared memory with where it leaved the file. Then it checks that did a new vaccine pair appeared after nurse added the new one or not. If a new vaccine pair (type-1, type-2) is appeared in clinic buffer then nurse increases the sem_underflow semaphore so one of the vaccinators can enter the clinic buffer and gets a pair of vaccines.

3.2 Vaccinators

Firstly lets see the content of the fifoname shared memory. Lets say we have 3 citizen and they must get 3 vaccines. So;

./F12321*3,./F23123*3,,./F23128*3,

As you can see, each citizen's name of fifo and vacines that citizen must get are separated by '*' character. And each citizen is seperated by ',' character. Fifo names are generated in form of ./F{pid of process} by the parent. So fifo files will be located at the location of program itself. And parent sorts the pid numbers in ascending order so oldest one is the first one.

I used a marking system to determine which fifo is available. If it is not available and a vaccinator is using it at the moment than fifoname has a '+' sign at the end of it. "/F12321*3*+," like that.

Lets say first vaccinator comes and looks at shared memory. It checks first element and it is './F12321*3'. So there is no '+' sign at the end of it, then it checks how many vaccines that citizen must get too. If citizen must get less or equal number of vaccines than previous one then it is unavailable. Because that means older one didn't get his/her vaccine so far. Lets give an example. If first one is "/F12321*3*+", then that is using by a vaccinator at the moment so it looks at next one which is "./F23123*3". So there is no "+" sign in it then it checks the number of vaccines of previous one which is "/F12321*3*+". That one must get 3 vaccines and so it can't pick "./F23123*3" one too even it has not "+" sign, since older one still didn't get his/her vaccine. If older one were "/F12321*2*+" or "/F12321*1*+" then it could pick "./F23123*3" one. Because older one got it's first or second vaccines at the moment. And vacinators checks all fifos like that. When it found a proper one than it marks it with "+" sign and parse the fifo name and vaccine number from it and stores them in variables to use later.

If all the fifo names are marked or all older ones has equal number of vaccines, then vaccinator uses sem_sending semaphore to suspend itself by decrease the value of semaphore. When a vaccinator applies vaccine to a citizen then that vaccinator removes the corresponding '+' mark from fifoname it used and increases the value of sem_sending. So one of the suspended vaccinators can awake and check the fifonames again.

After a vaccinator gets a fifoname from fifoname shared memory, it goes to clinic buffer. If there is a pair of vaccine then removes them, counts total number of vaccines in clinic buffer after removed 1 pair of them, stores vaccine type-1 and type-2 numbers in variables and increases the value of sem_overflow 2 times since it removes 2 vaccines from clinic. So 2 nurses can carry new vaccines from file to clinic buffer.

After it gets vaccines from buffer, it creates a message string like that "VAC1:12,VAC2:21,PID:12123". In this message VAC1 indicates the number of type1 vaccines in the clinic buffer when that vaccinator process removes them, VAC2 indicated the number of type2 vaccines and PID: indicates the pid of that vaccinator.

After it creates message it sends SIGUSR1 signal to citizen because each citizen suspends itself until gets SIGUSR1 signal. So citizen can know a message sent to it. Then vaccinator sends message to associated citizen via fifo and suspends itself for SIGUSR1 signal. The reason of suspending itself is it must make sure that citizen got the message. After citizen gets the message, that citizen sends SIGUSR1 signal to vaccinator, therefore vaccinator can know

citizen got the message. Citizen obtains that vaccinator's pid from the message.

As last step, that vaccinator access to fifoname shared memory and removes '+' sign at the end of associated fifoname and decreases the number of vaccines that citizen must get by 1 so any other vaccinator can pick that fifo and apply a new vaccine to that citizen. Lets say that vaccinator picked "./F12321*3" one. So it turned into "./F12321*3*+", after that vaccinator picked that one. Now it changes it into "./F12321*2". If number of vaccines that citizen must get is turns into 0 in this part, then vaccinator doesn't change it, instead it removes it from the shared memory because that citizen got all the vaccines he/she must get. If that is the last citizen and it got the last vaccine of his/her then vaccinator writes "GAMEFINISHED" into fifoname shared memory. When a vaccinator see "GAMEFINISHED" in that sharedmemory, then they breaks the main loop and exit the program properly.

When all citizens get their all vaccines, all vaccinators access to vaccinator shared memory one by one and writes how many vaccines they applied until now. So parent process can print those output at the end of the program.

3.3 Citizens

Citizens firstly suspends themself until a SIGUSR1 signal comes from a vaccinator. If it comes than thay know a message will come and they read the fifo. They get a message from the vaccinator that contains number of vaccine1 and vaccine2 in the clinic when that vaccinator removes them and pid number of vaccinator. Right after that, citizen sends SIGUSR1 signal back to vaccinator so that vaccinator can be notified that citizen got the message. With that communication I make sure that another vaccinator doesn't pick that citizen again and send the message again before that citizen reads the message that comes from previous vaccinator. After that part it decreases the number of vaccines he/she must get by 1 and prints the message like the pdf and goes back to loop again and suspends for SIGUSR1 again until another vaccinator awakes that citizen. If number of vaccines that citizen is decreased to 0, then instead of suspends itself it just break the loop and removes it's fifo and resources.

3.4 Cleaner

There is 1 process I created to make sure that there will not be suspended or blocked process due to reading fifo when SIGINT signal comes. That process from the very start suspends itself until SIGUSR1 or SIGINT signals comes. If program ends normally, some vaccinators gets cleaner's pid from shared memory and sends SIGUSR1 signals to that process and it exits the program normally without do anything special. If it awakes with signal of SIGINT then it increases all the semaphore values, sends SIGUSR1 signals to all other processes and writes a message to all the fifo, so it makes sure that there will be

no such process that is suspended or blocked due to reading a fifo after SIGINT comes.

4 IMPORTANT

Since my solution includes the parsing the fifonames from shared memory. If there is a huge number of citizens. Vaccinators has to spend more time on fifoname shared memory since it will contain a huge string. So at the beginning of the program it might seem like just nurses work.

5 OUTPUTS

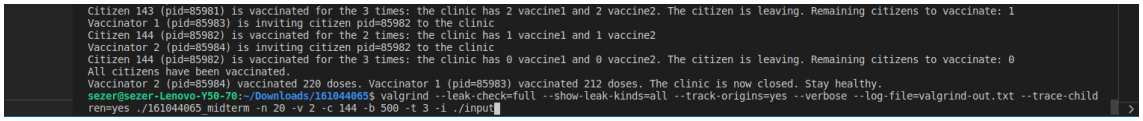

Figure 1: Input Example

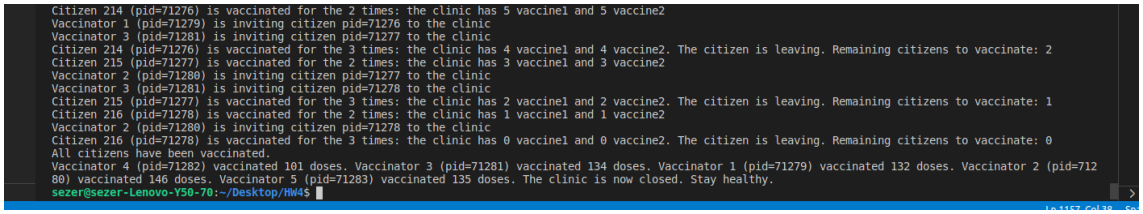

Figure 2: Output Example

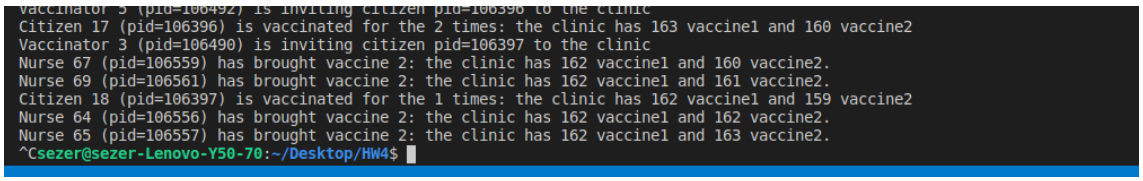

Figure 3: CTRL-C Interrupt Example