Gebze Technical University
Department of Computer Engineering

CSE 344 - MIDTERM REPORT

SEMAPHORES

Student:
Ahmet Melih YANALAK
Advisor:
Prof. Dr. Erchan APTOULA

02-05-2021

Contents

1 Definition Of The Project

The aim of this project is emulating a covid-19 vaccination flow. Nurses bring vaccines to a clinic, vaccinators get 2 vaccines at once (1 Vaccine1 and 1 Vaccine2) and invite citizens to the clinic. Each Nurse, Vaccinator and Citizen will be represented by a distinct process.

2 Visualizing the Structure

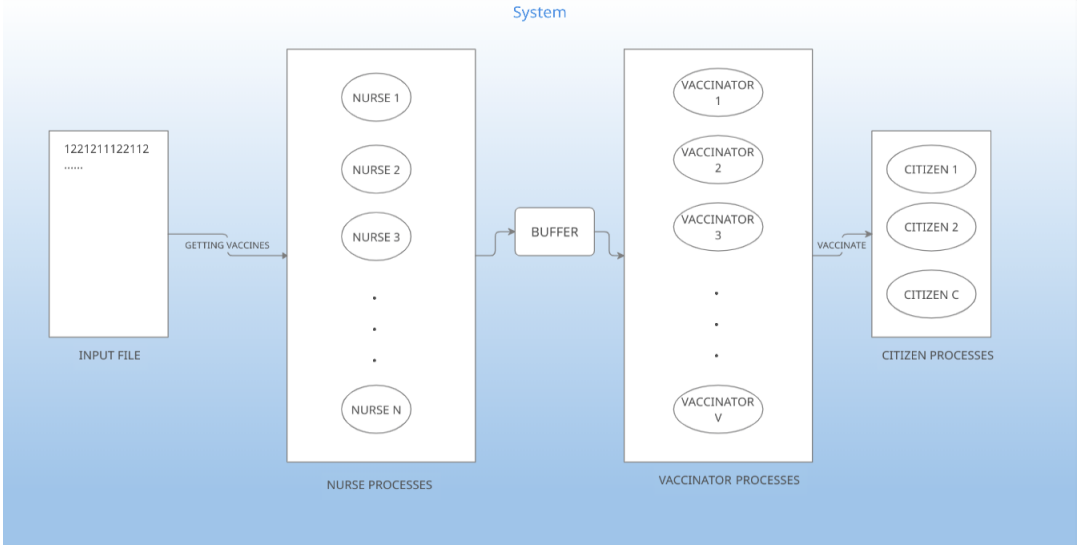

Figure 1.1: Structure of the program

In order to start the flow, Nurse processes will produce vaccines to clinic(buffer),meanwhile vaccinators taking the vaccines from clinic(buffer) and inviting the citizens if there are sufficient amount of vaccines.If there are not room in the clinic then nurse(producer) waits until vaccinators consuming the vaccines from clinic(buffer).

3 Starting the Program : Getting Input Values

In order for the program to run properly without having any trouble,the input parameters entered by user must be checked by using getopt library.

4 Processes

4.1 Nurse Process Design and Implementation

Nurse processes should read the ASCII Input file one byte at a time and add those values to buffer.After reading one character which can be either '1' or '2', the V1 or V2 value that is kept in shared memory will be increased.If the number of the vaccines in type that current character has was lower than the number of the vaccines in other type, then v1v2 semaphore will be posted.In other words if there are vaccine1 and vaccine2 in the buffer after nurse read the file,then it will be posted to v1v2 semaphore to increase the number. Termination condition of nurse process is reaching the end of file.

In order to avoid the condition where multiple nurses trying to read from file, a semaphore named "mutex" is used to lock the critical region.While nurse process in mutex region,other nurse processes and also vaccinator processes can not enter the critical region.After finishing the adding to buffer operation,mutex is unlocked to let other processes may run.

NOTE: This figure is not the latest version.It only used to explain the concept(Signal handling part is not shown yet for the sake of simplicity).

```
void nurse(int fd,int i){
  char buffer[1];
  while (1) {
    my_sem_wait(empty);
    my_sem_wait(mutex);
    if (read(fd,buffer,1) <= 0) {
      my_sem_post(mutex);
      break;
    }
    if (buffer[0] == '1') {
      if (bufferInfo->v1 < bufferInfo->v2 ) {
        my_sem_post(v1v2);
      }
      bufferInfo->v1 = bufferInfo->v1+1;
    }
    else if (buffer[0] == '2'){
      if (bufferInfo->v1 > bufferInfo->v2 ) {
        my_sem_post(v1v2);
      }
      bufferInfo->v2 = bufferInfo->v2 +1;
    }
    else{
        printf("Nurses have carried all vaccines to the buffer, terminating.\n");
        my_sem_post(mutex);
        my_sem_post(v1v2);
        return;
    }
    printf("Nurse %d (pid=%d) has brought vaccine %s: the clinic has %d vaccine 1 and %d vaccine 2.'
    my_sem_post(mutex);

  }
  return;
}
```

Figure 1.2: Implementation of the Nurse Process

4.2 Vaccinator Process Design and Implementation

Vaccinator processes should wait until there is enough vaccines(at least 1 from each) to invite citizens into clinic. The "v1v2" semaphore is used in order to provide the synchronization.If there are not sufficient number of vaccines then vaccine processes sleeping.After nurse providing both vaccine types and posting v1v2 semaphore, vaccinator processes will wait for mutex to be unlocked and enter the critical region.

In the critical region vaccinator process knows that it has 2 vaccines that citizens need,vaccinator invites the citizen which has the next turn(from oldest to youngest) by communicating via pipe. Each citizen has their own pipe in order to figure out if that process itself has the turn. A variable named "nextCitizen" is kept in the shared memory in order to decide which citizen to invite to clinic. This variable is increased every time after one of the citizen get vaccinated.In order to call the citizen process that has the turn, nextCitizen modulus with respect to number of Citizens will give the process to invite. Vaccinator write 1 byte to pipe of the citizen to be invited and triggers it.

NOTE: This figure is not the latest version.It only used to explain the concept(Signal handling part is not shown yet for the sake of simplicity).

```
int vaccinator(int i){
  int returnNum = 0;
  int isFinishedNum;
  //int v1Num,v2Num;
  char *temp = "1";
  vaccinatorInfo[(2*bufferInfo->numberOfVaccinators)-(i-1)-1] = getpid();
  while (1){
    my_sem_wait(v1v2);
    my_sem_wait(mutex);

    sem_getvalue(isFinished,&isFinishedNum);


    bufferInfo->v1 = bufferInfo->v1-1;
    bufferInfo->v2 = bufferInfo->v2-1;
    my_sem_post(empty);
    my_sem_post(empty);
    bufferInfo->totalConsumedSize = bufferInfo->totalConsumedSize +2;
    printf("Vaccinator %d (pid=%d) is inviting citizen pid=%d to the clinic.\n",i,getpid(),citizenInfo[bufferI

    close(pipeFd[bufferInfo->nextCitizen % bufferInfo->numberOfCitizens][0]);
    if(write(pipeFd[bufferInfo->nextCitizen % bufferInfo->numberOfCitizens][1], temp, 1)<0) {  //pipe'ın icine
      perror("write-- ");
    }
    my_sem_wait(citizenDoneSem);

    returnNum++;
```

Figure 1.3: Implementation of the Vaccinator Process

4.3 Citizen Process Design and Implementation (Bonus Part)

Citizens are processes that waits to be called by a vaccinator process. Each citizen needs to be vaccinated t times.After a citizen gets vaccinated t times, citizen process terminates after returning recources.

All citizens must wait until their turn is next(From oldest to youngest).In order to handle this synchronization, pipe is used. Each citizen has their own pipe to communicate with vaccinators. Citizens waiting to read 1 byte from that pipe and sleeps until its written which means that citizens get their vaccines in the order of age.Each round citizens get vaccinated from oldest to youngest citizen and after finishing round,the oldest one gets vaccinated again.

Note: On Linux, more than around 500 pipes are not allowed.For this reason, input citizen number should be less than 500.

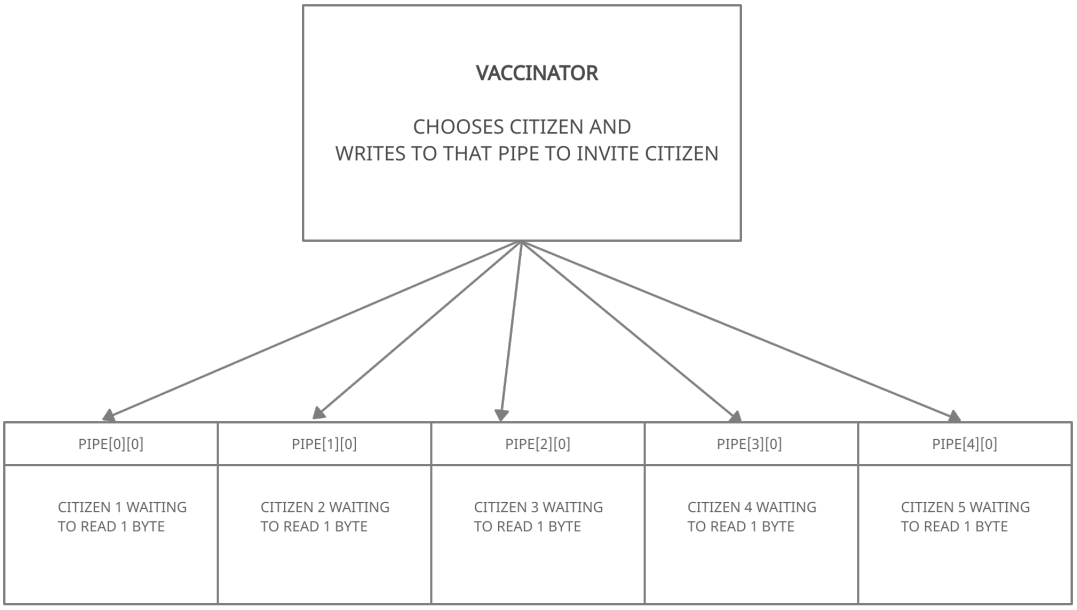

Figure 1.4: Design of Pipe

In order for citizens to communicate with vaccinators, vaccinators writes to pipe and waits for a semaphore called "citizenDone". After getting vaccinated, each citizen post that semaphore.It provides protection while a citizen getting vaccinated.

NOTE: This figure is not the latest version.It only used to explain the concept(Signal handling part is not shown yet for the sake of simplicity).

```
void citizen(int citizenNumber,int t){
  int i = 1;
  pid_t pid;
  char temp[1];
  pid = getpid();
  while (i <= t) {

    close(pipeFd[citizenNumber-1][1]);

    if(read(pipeFd[citizenNumber-1][0], temp,1)<0) {
      perror("readPipe ");
    }
    if (i == t) {
      bufferInfo->remainingCitizen = bufferInfo->remainingCitizen -1 ;
      printf("Citizen %d (pid=%d) is vaccinated for the %d time: the clinic has %d vaccine 1 and %d vaccine
    }
    else{
      printf("Citizen %d (pid=%d) is vaccinated for the %d time: the clinic has %d vaccine 1 and %d vaccine
    }
    bufferInfo->nextCitizen = bufferInfo->nextCitizen + 1;
    my_sem_post(citizenDoneSem);
    i++;


  }
  close(pipeFd[citizenNumber-1][0]);
```

Figure 1.5: Implementation of Citizen Process

5 Semaphores and Shared Memory

5.1 Semaphores

In order for to provide synchronization between processes, 5 POSIX Semaphores are used in this project:

• MUTEX

Mutex is used to lock the critical region in order to prevent existence of multiple processes in the critical region simultaneously.

• EMPTY

Empty Semaphore is used to prevent the nurse processes to enter the critical region if there is not enough room in the clinic(buffer).In this case, nurse processes should wait until vaccinator processes consume the vaccines.

• V1V2

v1v2 semaphore is used to count the pair of v1,v2 vaccines available.If there is not sufficient number of vaccines in the buffer,vaccinator processes should wait until nurse processes add vaccines to clinic(buffer).

• CITIZENDONE

CitizenDone semaphore is used to block vaccinators to invite other citizens while current citizen is getting vaccinated. The Vaccinator who invited the citizen waits for this semaphore until that citizen is finished. After a citizen finished vaccinating, citizen post that semaphore to let vaccinator leave the critical region nicely.

• ISFINISHED

IsFinished semaphore is used to terminate the vaccinator processes nicely. When a vaccinator process finished, others may be still waiting v1v2 semaphore and since vaccines are finished there will be deadlock.In order to prevent this situation,after a vaccinator process finished, it triggers other vaccinator processes to terminate also.

5.2 Shared Memory

There are 3 shared memory used in this project:

• First one keeps a list of Process ID's of Citizens to be used while printing which citizen to invite in vaccinator process.

• Second one keeps a list of Process ID's and Vaccinate numbers of each Vaccinators in order to print at the end of the program.

• Third one keeps a struct which have information fields that is used in the execution of the program.

Fields are:

Field	Used For
totalFileSize	To Check if all vaccines carried
totalConsumedSize	Keeps current consumed vaccines
remainingCitizen	How many citizen process still running
numberOfVaccinators	Number of Vaccinators
nextCitizen	Next Citizen has priority to get vaccinated
numberOfCitizens	Citizen number is used to get modulus of nextCitizen
v1	Number of Vaccine1
v2	Number of Vaccine2

Table 1.1: Shared Memory Fields

6 Signal Handler

When the user press CTRL-C while program is still running , in order to terminate the program gracefully all controls are added to process functions(nurse, citizen, vaccinator).
An atomic global variable "exitThread" is used to check the condition of CTRL-C.If user pressed CTRL-C then void handler starts running and it makes that variable '1' which is terminating condition for all loops in process functions.
In addition, in order to prevent any process waiting for a semaphore and getting deadlock situation,processes post semaphores to trigger each other so that they can break the loop and terminate without printing or having any memory leak.

```
void signalhandler(int signum, siginfo_t *info, void *ptr)
{

    if (signum == SIGINT)
    {

        printf("\n\n-USER PRESSED CTRL-C,PROGRAM TERMINATING GRACEFULLY-\n\n");

        if (exitThread != -1)
        {
            exitThread = -1;
        }
    }
}
```

Figure 1.6: Signal Handler

```
if (exitThread == -1) {
  for (int i = 0; i < bufferInfo->numberOfCitizens; i++) {
    close(pipeFd[i][0]);
    if(write(pipeFd[i][1], temp, 1)<0) {
      perror("write-- ");
    }

  }
  my_sem_post(empty);
  my_sem_post(mutex);
}
```

Figure 1.7: Vaccinator CTRL-C Handler

7 Running Results

7.1 Test Case 1

The program is tested with many different input files and parameters.
In first test case input file is like this:

111111111111222222222222
which consist of 1's in first 12 bytes and 2's in last 12 bytes.
When program started with arguments : ./program -n 3 -v 3 -c 4 -b 13 -t 3 -i inputFile
Then the output of the program is below:

```
melo@Melo:~/Desktop/MIDTERM$ ./program -n 3 -v 3 -c 4 -b 13 -t 3 -i little
Welcome to the GTU344 clinic. Number of citizens to vaccinate c=4 with t=3 doses.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 1 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 2 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 3 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 4 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 5 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 6 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 7 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 8 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 9 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 10 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 11 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 1: the clinic has 12 vaccine 1 and 0 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 2: the clinic has 12 vaccine 1 and 1 vaccine 2.
Vaccinator 1 (pid=19090) is inviting citizen pid=19086 to the clinic.
Citizen 1 (pid=19086) is vaccinated for the 1 time: the clinic has 11 vaccine 1 and 0 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 11 vaccine 1 and 1 vaccine 2.
Nurse 1 (pid=19083) has brought vaccine 2: the clinic has 11 vaccine 1 and 2 vaccine 2.
Vaccinator 2 (pid=19091) is inviting citizen pid=19087 to the clinic.
Citizen 2 (pid=19087) is vaccinated for the 1 time: the clinic has 10 vaccine 1 and 1 vaccine 2.
Vaccinator 3 (pid=19092) is inviting citizen pid=19088 to the clinic.
Citizen 3 (pid=19088) is vaccinated for the 1 time: the clinic has 9 vaccine 1 and 0 vaccine 2.
Nurse 3 (pid=19085) has brought vaccine 2: the clinic has 9 vaccine 1 and 1 vaccine 2.
Nurse 3 (pid=19085) has brought vaccine 2: the clinic has 9 vaccine 1 and 2 vaccine 2.
Vaccinator 2 (pid=19091) is inviting citizen pid=19089 to the clinic.
Citizen 4 (pid=19089) is vaccinated for the 1 time: the clinic has 8 vaccine 1 and 1 vaccine 2.
Vaccinator 1 (pid=19090) is inviting citizen pid=19086 to the clinic.
Citizen 1 (pid=19086) is vaccinated for the 2 time: the clinic has 7 vaccine 1 and 0 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 7 vaccine 1 and 1 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 7 vaccine 1 and 2 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 7 vaccine 1 and 3 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 7 vaccine 1 and 4 vaccine 2.
Nurse 3 (pid=19085) has brought vaccine 2: the clinic has 7 vaccine 1 and 5 vaccine 2.
Vaccinator 2 (pid=19091) is inviting citizen pid=19087 to the clinic.
Citizen 2 (pid=19087) is vaccinated for the 2 time: the clinic has 6 vaccine 1 and 4 vaccine 2.
Vaccinator 2 (pid=19091) is inviting citizen pid=19088 to the clinic.
Citizen 3 (pid=19088) is vaccinated for the 2 time: the clinic has 5 vaccine 1 and 3 vaccine 2.
Vaccinator 2 (pid=19091) is inviting citizen pid=19089 to the clinic.
Citizen 4 (pid=19089) is vaccinated for the 2 time: the clinic has 4 vaccine 1 and 2 vaccine 2.
Vaccinator 3 (pid=19092) is inviting citizen pid=19086 to the clinic.
Citizen 1 (pid=19086) is vaccinated for the 3 time: the clinic has 3 vaccine 1 and 1 vaccine 2. The citizen is leaving. Remaining citizens to vaccinate: 3
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 3 vaccine 1 and 2 vaccine 2.
Nurse 2 (pid=19084) has brought vaccine 2: the clinic has 3 vaccine 1 and 3 vaccine 2.
Nurses have carried all vaccines to the buffer, terminating.
Vaccinator 1 (pid=19090) is inviting citizen pid=19087 to the clinic.
Citizen 2 (pid=19087) is vaccinated for the 3 time: the clinic has 2 vaccine 1 and 2 vaccine 2. The citizen is leaving. Remaining citizens to vaccinate: 2
Vaccinator 1 (pid=19090) is inviting citizen pid=19088 to the clinic.
Citizen 3 (pid=19088) is vaccinated for the 3 time: the clinic has 1 vaccine 1 and 1 vaccine 2. The citizen is leaving. Remaining citizens to vaccinate: 1
Vaccinator 3 (pid=19092) is inviting citizen pid=19089 to the clinic.
Citizen 4 (pid=19089) is vaccinated for the 3 time: the clinic has 0 vaccine 1 and 0 vaccine 2. The citizen is leaving. Remaining citizens to vaccinate: 0
All citizens have been vaccinated .
Vaccinator 1 (pid=19090) vaccinated 4 doses. Vaccinator 2 (pid=19091) vaccinated 5 doses. Vaccinator 3 (pid=19092) vaccinated 3 doses. The clinic is now closed. Stay healthy.
```

Figure 1.8: Example Output

7.2 Test Case 2

For the second example input file is like this:

11112122122112122211

When program started with arguments : ./program -n 3 -v 3 -c 5 -b 13 -t 2 -i little

Then the output of the program is below:

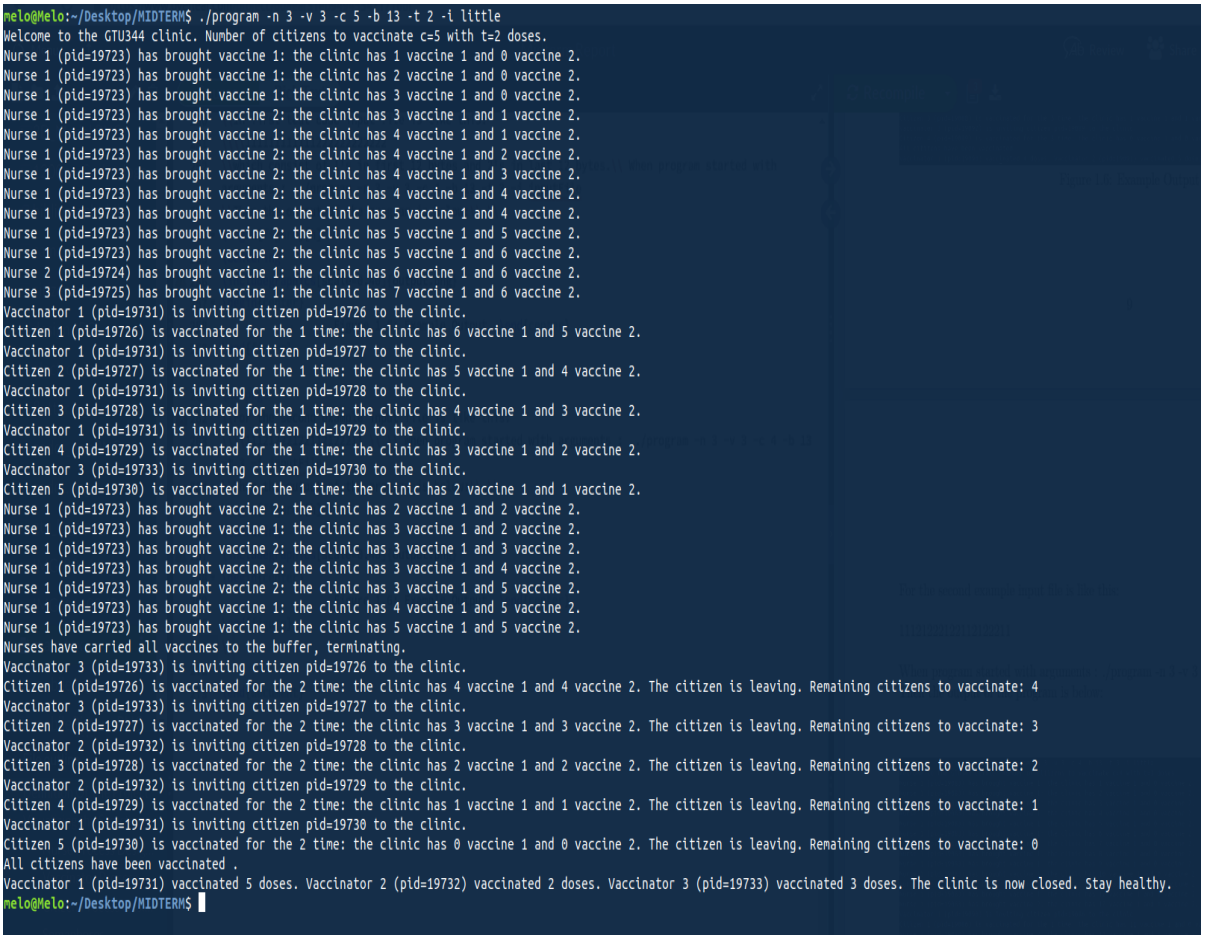

Figure 1.9: Example Output 2