

Midterm

Muhammed Yasir Fidan

May 2021

1 Report

Firstly, I use 7 posix named semaphores in my program. I hold my buffer as a shared memory. Empty semaphore and mshare semaphore same as producer consumer program semaphores that we saw in class. I use empty semaphore in nurse processes. Empty semaphore check there is still empty space in buffer or not. Empty semaphore initialize as buffer size at the beginning. If it become 0 all nurses wait for some vaccinator and citizens use vaccines and make empty spaces in buffer. Then Nurses can continue to fill buffer.

The mshare semaphore again very similar to producer consumer semaphore. I use it to lock whenever I access shared memory. Because I don't want to more than 1 process can access shared memory at the same time. If they can reach at the same time, They can corrupt memory so mshare used for critical section locks.

Figure 1 is contains my nurse process code. All process read 1 character from file. I use exclusive lock when reading char because I don't want to 2 process read at the same time. Maybe they can read same character if they read at the same time so when reading I use exclusive lock. After processes read 1 character they use empty semaphore to check there is empty space in buffer or not. If there is empty space processes continue. Then only one process enter mshare semaphore critical region like in producer consumer problem. Kernel choose one of the nurse process and it enter mshare critical area. This is critical area because I access my buffer here. If nurse has a vaccine 1 insertBuffer function add a vaccine 1 to the buffer. If nurse has a vaccine 2 insertBuffer function add a vaccine 2 to the buffer. Then I take how many vaccine 1 and vaccine 2 in my buffer by using numberOfVaccine1 and numberOfVaccine2 functions. Then I use 2 more semaphore here. removeVac and atomicWait. atomicWait used in vaccinators. All vaccinators wait for there is at least 1 vaccine 1 and 1 vaccine 2 in buffer. In nurse, here I check there is currently enough vaccines for a new vaccinator to run by using removeVac and atomic. If there is both 1 and 2 vaccine in buffer now a vaccinator that wait in atomicWait semaphore now can continue. After that nurse release mshare lock. Now kernel continue with same nurse or other nurses same process again until nurses reach end of file.

```

205 if(nurse[i] == 0){ // Nurse code
206     int count = 1;
207     char chr;
208     //lock for file
209     struct flock lock;
210     memset(&lock,0,sizeof(lock));
211     while(1){
212         //Exclusive lock file to read 1 char
213         lock.l_type = F_WRLCK;
214         if(fcntl(fd,F_SETLKW,&lock) == -1){
215             perror("fcntl error");
216             exit(1);
217         }
218         count = read(fd,&chr,sizeof(chr));
219         if(count == -1){
220             perror("Read error");
221             exit(1);
222         }
223         //After reading 1 char open lock so other process can access file
224         lock.l_type = F_UNLCK;
225         if(fcntl(fd,F_SETLK,&lock) == -1){
226             perror("fcntl error");
227             exit(1);
228         }
229         //Check ctrl+c signal
230         if(flag > 0){
231             close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);
232             exit(0);
233         }
234     }
235
236     if(count == 0) break; //EOF
237
238     if(chr != '\n'){
239         //accessing shared memory lock
240         sem_wait(empty);
241         sem_wait(mshare);
242         insert_buffer(addr,chr);
243
244         int valuem1,valuem2;
245         valuem1 = number_of_vaccine_1(addr);
246         valuem2 = number_of_vaccine_2(addr);
247         printf("Nurse %d (pid=%ld) has brought vaccine %c: the clinic has %d vaccine1 and %d vaccine2.\n",
248             i+1,(long)getpid(),chr,valuem1,valuem2);
249
250         int x;
251         sem_getvalue(removeVac,&x);
252         if(valuem1-x > 0 && valuem2-x > 0){
253             sem_post(atomic_wait);
254             sem_post(removeVac);
255         }
256         sem_post(mshare);

```

Figure 1: The Nurse process code

Figure 2 contains my vaccinator process code. In vaccinator first I check in critical section there is enough vaccinator for my program already or not. Because file will contain $2*t*c$ character so vaccinators work for $t*c$ times total. So I check it at the beginning of vaccinator code. If vaccinator already executed for $t*c$ times it means all citizens already finished. So vaccinators stop in if statement and they start exit too after closing all semaphores.

But this is the case for exit of vaccinators. If there are still citizens must invite, they continue simultaneously and stuck in atomicWait semaphore. They wait here for at least there will be 1 vaccine1 and 1 vaccine2 in buffer. So I explained in above nurses increment this semaphore when ever there is enough vaccines for a vaccinator. So kernel choose one of them and this vaccinator can continue and others wait for more vaccines. Then one of the vaccinator enter critical region because I modify shared memory here. First Vaccinator decrease removeVac semaphore and call remove1and2 function. This function removes 1 vaccine1 and 1 vaccine2 in buffer. Then I use my fifth semaphore civ. Vaccinator post

civ semaphore so 1 of the citizen process awakened. Then I construct a fifo between this vaccinator and awakened citizen because I need citizen pid. After reading citizen pid from fifo, Vaccinator print invite message something in pdf. After that I use my second shared memory. I use this memory to hold which vaccinator invite how many people information. So I increment 1 for this vaccinator because he/she invite a citizen. Then I use my last 2 semaphore those are m1 and m2. I used this semaphores for ensure first write invite message than citizen vaccinated message. After that Vaccinator release lock and increment empty semaphore 2 times. Because he/she use 2 vaccine.

```

277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327

```

```

while(1){
    sem_wait(mshare);
    int number_of_vac = take_number_of_vaccinator(third_addr);
    if( number_of_vac != t*c) incr_number_of_vaccinator(third_addr);
    sem_post(mshare);

    if( number_of_vac == t*c){
        //close semaphores
        close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);

        exit(0);
    }

    sem_wait(atomic_wait);

    //critical section
    //update shared memory
    sem_wait(mshare);

    sem_wait(removeVac);
    remove_land2(addr);

    sem_post(civ); // take a civ

    int fifoFd;
    while(((fifoFd = open("fifo1",O_RDONLY)) == -1) && (errno == EINTR));
    if(fifoFd == -1){
        perror("open fifo error");
        exit(1);
    }
    char read_civ_pid[10];
    int checkReadFail = read(fifoFd,read_civ_pid,sizeof(read_civ_pid));
    if(checkReadFail == -1){
        perror("Fifo Read Error in vaccinator");
        exit(1);
    }
    if(close(fifoFd) == -1){
        perror("Close error in vaccinator");
        exit(1);
    }

    printf("Vaccinator %d (pid=%ld) is inviting citizen pid=%s\n",i+1,(long) getpid(),read_civ_pid);
    incr_vaccine_count(i+1,helperAddr);

    sem_post(m1);
    sem_wait(m2);
    sem_post(mshare);
}

```

Line 293, Column 1

Figure 2: The vaccinator process code

```

344
345 //create citizen
346 for(int i=0; i<c; i++){
347     citizen[i] = fork();
348     //check fork error
349     if(citizen[i] == -1){
350         perror("Error in citizen fork");
351         exit(1);
352     }
353
354     if(citizen[i] == 0){ // citizen code
355         int cit_pid = (int) getpid();
356         char char_id[20];
357         for(int j=0; j<t; j++){
358             sem_wait(civ);
359
360             snprintf(char_id,10,"%d",cit_pid);
361
362             int fifoFd;
363             while(((fifoFd = open("fifo1",O_WRONLY)) == -1) && (errno == EINTR));
364             if(fifoFd == -1){
365                 perror("open fifo error");
366                 exit(1);
367             }
368             int checkWriteFail = write(fifoFd,char_id,sizeof(char_id));
369             if(checkWriteFail == -1){
370                 perror("Write error in citizen");
371                 exit(1);
372             }
373             if(close(fifoFd) == -1){
374                 perror("Close error in citizen");
375                 exit(1);
376             }
377
378             sem_wait(m1);
379             printf("Citizen %d (pid=%ld) is vaccinated for the %dth time: the clinic has %d vaccine1 and %d vaccine2\n",
380                 i,(long)getpid(),j+1,number_of_vaccine_1(addr),number_of_vaccine_2(addr));
381             if(j+1 == t){
382                 int rem_cit = remaining_citizen(helperAddr);
383                 printf("Citizen is leaving. Remaining citizens to vaccinate: %d\n", rem_cit-1);
384                 decr_remaining_citizen(helperAddr);
385                 if(rem_cit - 1 == 0){
386                     printf("All Citizens have been vaccinated.\n");
387                 }
388             }
389             sem_post(m2);
390             //Check ctrl+C Signal
391             if(flag > 0){
392                 close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);
393             }

```

Line 347, Column 29

Figure 3: The citizen process code

Citizen wait in `semwait(civ)` for a vaccinator awake them. Then awakened citizen use `fifo` that we opened in vaccinator and send its own `pid` to vaccinator. Then wait `m1` semaphore wait for vaccinator print invite message then citizen can continue and print its message. Also this shared memory used in citizen for printing remaining citizens information.

Ctrl C signal

In case of `ctrl c` signal I use a flag in signal handler. Child processes check this flag in their code. If a signal arrived at least one of the child terminated because of the `ctrl c` signal. Then parent will kill all children and print a signal catch message and terminate after closing all shared memory and semaphores.

In my program design, Vaccinator block another invite before a citizen take its vaccine. So in my output, After every vaccinator invite message there is citizen vaccinated message.

Output example

`./program -n 10 -v 4 -c 6 -b 40 -t 6 -i inputFile`

```
cse312@ubuntu: ~/Desktop/system/System-Programming/midterm
cse312@ubuntu:~/Desktop/system/System-Programming/midterm$ ./program -n 10 -v 4 -c 6 -b 40 -t 6 -i inputFile
Welcome to the CSE344 clinic. Number of citizens to vaccinate c=6 with t=6 doses.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 1 vaccine1 and 0 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 2 vaccine1 and 0 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 2 vaccine1 and 1 vaccine2.
Nurse 4 (pid=15576) has brought vaccine 2: the clinic has 2 vaccine1 and 2 vaccine2.
Nurse 3 (pid=15575) has brought vaccine 2: the clinic has 2 vaccine1 and 3 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 3 vaccine1 and 3 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 3 vaccine1 and 4 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 4 vaccine1 and 4 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 4 vaccine1 and 5 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 5 vaccine1 and 5 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 5 vaccine1 and 6 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 6 vaccine1 and 6 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 7 vaccine1 and 6 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 7 vaccine1 and 7 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 7 vaccine1 and 8 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 7 vaccine1 and 9 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 8 vaccine1 and 9 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 9 vaccine1 and 9 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 9 vaccine1 and 10 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 9 vaccine1 and 11 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 9 vaccine1 and 12 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 10 vaccine1 and 12 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 10 vaccine1 and 13 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 10 vaccine1 and 14 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 11 vaccine1 and 14 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 12 vaccine1 and 14 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 12 vaccine1 and 15 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 13 vaccine1 and 15 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 14 vaccine1 and 15 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 14 vaccine1 and 16 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 14 vaccine1 and 17 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 1: the clinic has 15 vaccine1 and 17 vaccine2.
Nurse 6 (pid=15578) has brought vaccine 1: the clinic has 16 vaccine1 and 17 vaccine2.
Nurse 4 (pid=15576) has brought vaccine 1: the clinic has 17 vaccine1 and 17 vaccine2.
Vaccinator 1 (pid=15583) is inviting citizen pid=15587
Citizen 0 (pid=15587) is vaccinated for the 1th time: the clinic has 16 vaccine1 and 16 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15588
Citizen 1 (pid=15588) is vaccinated for the 1th time: the clinic has 15 vaccine1 and 15 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15589
Citizen 2 (pid=15589) is vaccinated for the 1th time: the clinic has 14 vaccine1 and 14 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15590
Citizen 3 (pid=15590) is vaccinated for the 1th time: the clinic has 13 vaccine1 and 13 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15591
Citizen 4 (pid=15591) is vaccinated for the 1th time: the clinic has 12 vaccine1 and 12 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15592
Citizen 5 (pid=15592) is vaccinated for the 1th time: the clinic has 11 vaccine1 and 11 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15587
Citizen 0 (pid=15587) is vaccinated for the 2th time: the clinic has 10 vaccine1 and 10 vaccine2
Vaccinator 1 (pid=15583) is inviting citizen pid=15588
```

Figure 4: Running start

Valgrind example shows no memory leak.

```

cse312@ubuntu: ~/Desktop/system/System-Programming/midterm
Vaccinator 3 (pid=15585) is inviting citizen pid=15590
Citizen 3 (pid=15590) is vaccinated for the 4th time: the clinic has 9 vaccine1 and 11 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15591
Citizen 4 (pid=15591) is vaccinated for the 4th time: the clinic has 8 vaccine1 and 10 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15592
Citizen 5 (pid=15592) is vaccinated for the 4th time: the clinic has 7 vaccine1 and 9 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15587
Citizen 0 (pid=15587) is vaccinated for the 5th time: the clinic has 6 vaccine1 and 8 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15588
Citizen 1 (pid=15588) is vaccinated for the 5th time: the clinic has 5 vaccine1 and 7 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15589
Citizen 2 (pid=15589) is vaccinated for the 5th time: the clinic has 4 vaccine1 and 6 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15590
Citizen 3 (pid=15590) is vaccinated for the 5th time: the clinic has 3 vaccine1 and 5 vaccine2
Vaccinator 3 (pid=15585) is inviting citizen pid=15591
Citizen 4 (pid=15591) is vaccinated for the 5th time: the clinic has 2 vaccine1 and 4 vaccine2
Nurse 4 (pid=15576) has brought vaccine 1: the clinic has 3 vaccine1 and 4 vaccine2.
Nurse 9 (pid=15581) has brought vaccine 1: the clinic has 4 vaccine1 and 4 vaccine2.
Nurse 5 (pid=15577) has brought vaccine 2: the clinic has 4 vaccine1 and 5 vaccine2.
Nurse 7 (pid=15579) has brought vaccine 2: the clinic has 4 vaccine1 and 6 vaccine2.
Nurse 10 (pid=15582) has brought vaccine 1: the clinic has 5 vaccine1 and 6 vaccine2.
Nurse 8 (pid=15580) has brought vaccine 1: the clinic has 6 vaccine1 and 6 vaccine2.
Vaccinator 2 (pid=15584) is inviting citizen pid=15592
Citizen 5 (pid=15592) is vaccinated for the 5th time: the clinic has 5 vaccine1 and 5 vaccine2
Vaccinator 2 (pid=15584) is inviting citizen pid=15587
Citizen 0 (pid=15587) is vaccinated for the 6th time: the clinic has 4 vaccine1 and 4 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 5
Vaccinator 2 (pid=15584) is inviting citizen pid=15588
Citizen 1 (pid=15588) is vaccinated for the 6th time: the clinic has 3 vaccine1 and 3 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 4
Vaccinator 2 (pid=15584) is inviting citizen pid=15589
Citizen 2 (pid=15589) is vaccinated for the 6th time: the clinic has 2 vaccine1 and 2 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 3
Nurse 6 (pid=15578) has brought vaccine 1: the clinic has 3 vaccine1 and 2 vaccine2.
Nurse 1 (pid=15573) has brought vaccine 2: the clinic has 3 vaccine1 and 3 vaccine2.
Nurses have carried all vaccines to the buffer, terminating.
Vaccinator 4 (pid=15586) is inviting citizen pid=15590
Citizen 3 (pid=15590) is vaccinated for the 6th time: the clinic has 2 vaccine1 and 2 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 2
Vaccinator 1 (pid=15583) is inviting citizen pid=15591
Citizen 4 (pid=15591) is vaccinated for the 6th time: the clinic has 1 vaccine1 and 1 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 1
Vaccinator 3 (pid=15585) is inviting citizen pid=15592
Citizen 5 (pid=15592) is vaccinated for the 6th time: the clinic has 0 vaccine1 and 0 vaccine2
Citizen is leaving. Remaining citizens to vaccinate: 0
All Citizens have been vaccinated.
Vaccinator 0 (pid=15583) vaccinated 18 doses.
Vaccinator 1 (pid=15584) vaccinated 4 doses.
Vaccinator 2 (pid=15585) vaccinated 13 doses.
Vaccinator 3 (pid=15586) vaccinated 1 doses.
cse312@ubuntu:~/Desktop/system/System-Programming/midterm$

```

Figure 5: Running end

```
valgrind-out.txt (-/Desktop/system/System-Programming/midterm) - gedit
inputFile x valgrind-out.txt x
--15697-- REDIR: 0x41ae900 (libc.so.6:__strcpy_ssse3) redirected to 0x402e5b0 (strcpy)
--15697-- REDIR: 0x40eb7b0 (libc.so.6:strcpy) redirected to 0x4024580 (_vgnU_ifunc_wrapper)
--15697-- REDIR: 0x40f4190 (libc.so.6:__strcpy_ssse3) redirected to 0x402d4c0 (strcpy)
==15687== HEAP SUMMARY:
==15687==    in use at exit: 0 bytes in 0 blocks
==15687==    total heap usage: 14 allocs, 14 frees, 297 bytes allocated
==15687== All heap blocks were freed -- no leaks are possible
==15687== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
--15694-- REDIR: 0x40e7de0 (libc.so.6:free) redirected to 0x402b370 (free)
==15686== HEAP SUMMARY:
==15686==    in use at exit: 0 bytes in 0 blocks
==15686==    total heap usage: 14 allocs, 14 frees, 297 bytes allocated
==15686== All heap blocks were freed -- no leaks are possible
==15686== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
--15698-- REDIR: 0x40ebf80 (libc.so.6:strncat) redirected to 0x4024580 (_vgnU_ifunc_wrapper)
--15698-- REDIR: 0x4101a50 (libc.so.6:__strncat_ssse3) redirected to 0x402d260 (strncat)
--15696-- REDIR: 0x40eb6e0 (libc.so.6:strcmp) redirected to 0x4024580 (_vgnU_ifunc_wrapper)
==15679== HEAP SUMMARY:
==15679==    in use at exit: 0 bytes in 0 blocks
==15679==    total heap usage: 14 allocs, 14 frees, 297 bytes allocated
==15679== All heap blocks were freed -- no leaks are possible
==15679== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==15688== HEAP SUMMARY:
==15688==    in use at exit: 0 bytes in 0 blocks
==15688==    total heap usage: 14 allocs, 14 frees, 297 bytes allocated
==15688== All heap blocks were freed -- no leaks are possible
==15688== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
--15696-- REDIR: 0x41aa900 (libc.so.6:__strcmp_ssse3) redirected to 0x402e5b0 (strcmp)
==15680== HEAP SUMMARY:
==15680==    in use at exit: 0 bytes in 0 blocks
```

Figure 6: Valgrind output memory leak