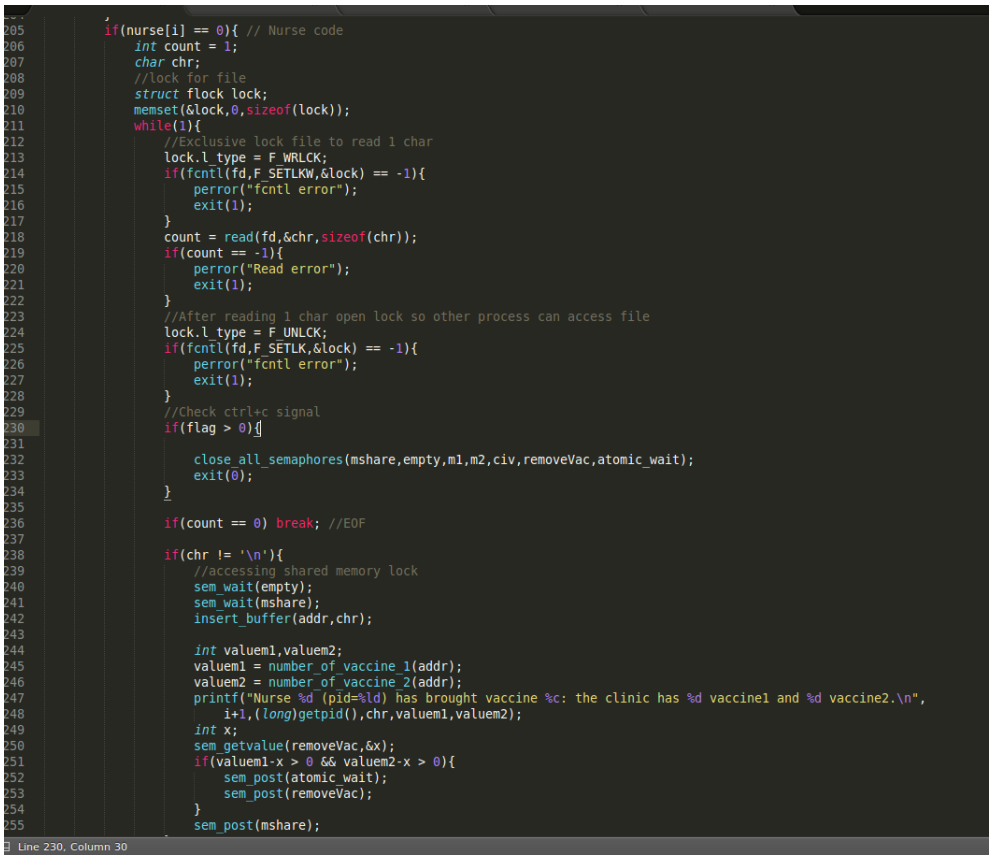
Midterm

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 comsumer 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 initilize as buffer size at the beginning. If it become 0 all nurses wait for some vaccinator and citizens use vaccines and make empty spaceses 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 containe 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 mshara critic 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 relase 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 
232                     close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);
233                     exit(0);
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                     int x;
250                     sem_getvalue(removeVac,&x);
251                     if(valuem1-x > 0 && valuem2-x > 0){
252                         sem_post(atomic_wait);
253                         sem_post(removeVac);
254                     }
255                     sem_post(mshare);
Line 230, Column 30
```

Figure 1: The Nurse process code

Figure 2 containe 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 containe 2*t*c character so vaccinators work for t*c times total. So I check it at the begginning 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 simultaniously 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 invete 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             while(1){
279
280                 sem_wait(mshare);
281                 int number_of_vac = take_number_of_vaccinator(third_addr);
282                 if( number_of_vac != t*c) incr_number_of_vaccinator(third_addr);
283                 sem_post(mshare);
284
285                 if( number_of_vac == t*c){
286                     //close semaphores
287                     close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);
288
289                     exit(0);
290                 }
291
292                 sem_wait(atomic_wait);
293
294                 //critical section
295                 //update shared memory
296                 sem_wait(mshare);
297
298                 sem_wait(removeVac);
299                 remove_land2(addr);
300
301                 sem_post(civ); // take a civ
302
303
304                 int fifoFd;
305                 while(((fifoFd = open("fifo1",O_RDONLY)) == -1) && (errno == EINTR));
306                 if(fifoFd == -1){
307                     perror("open fifo error");
308                     exit(1);
309                 }
310                 char read_civ_pid[10];
311                 int checkReadFail = read(fifoFd,read_civ_pid,sizeof(read_civ_pid));
312                 if(checkReadFail == -1){
313                     perror("Fifo Read Error in vaccinator");
314                     exit(1);
315                 }
316                 if(close(fifoFd) == -1){
317                     perror("Close error in vaccinator");
318                     exit(1);
319                 }
320
321
322                 printf("Vaccinator %d (pid=%ld) is inviting citizen pid=%s\n",i+1,(long) getpid(),read_civ_pid);
323                 incr_vaccine_count(i+1,helperAddr);
324
325                 sem_post(m1);
326                 sem_wait(m2);
327                 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
379                 sem_wait(m1);
380                 printf("Citizen %d (pid=%ld) is vaccinated for the %dth time: the clinic has %d vaccine1 and %d vaccine2\n",
381                     i,(long)getpid(),j+1,number_of_vaccine_1(addr),number_of_vaccine_2(addr));
382                 if(j+1 == t){
383                     int rem_cit = remaining_citizen(helperAddr);
384                     printf("Citizen is leaving. Remaining citizens to vaccinate: %d\n", rem_cit-1);
385                     decr_remaining_citizen(helperAddr);
386                     if(rem_cit - 1 == 0){
387                         printf("All Citizens have been vaccinated.\n");
388                     }
389                 }
390                 sem_post(m2);
391                 //Check ctrl+C Signal
392                 if(flag > 0){
393                     close_all_semaphores(mshare,empty,m1,m2,civ,removeVac,atomic_wait);
Line 347, Column 29
```

Figure 3: The citizen process code

Citizen wait in semwait(civ) for a vaccinator aweke 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 thirs 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

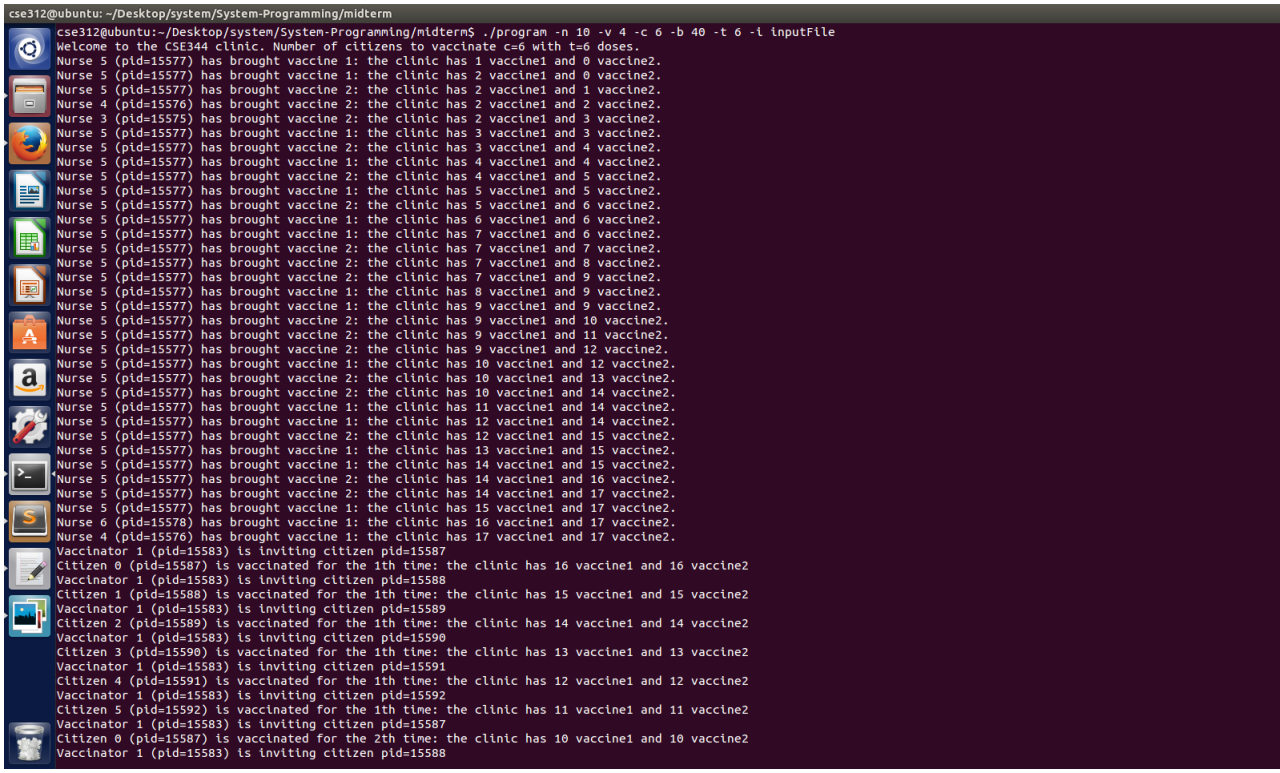


Figure 4: Running start

Valgrind example shows no memory leak.

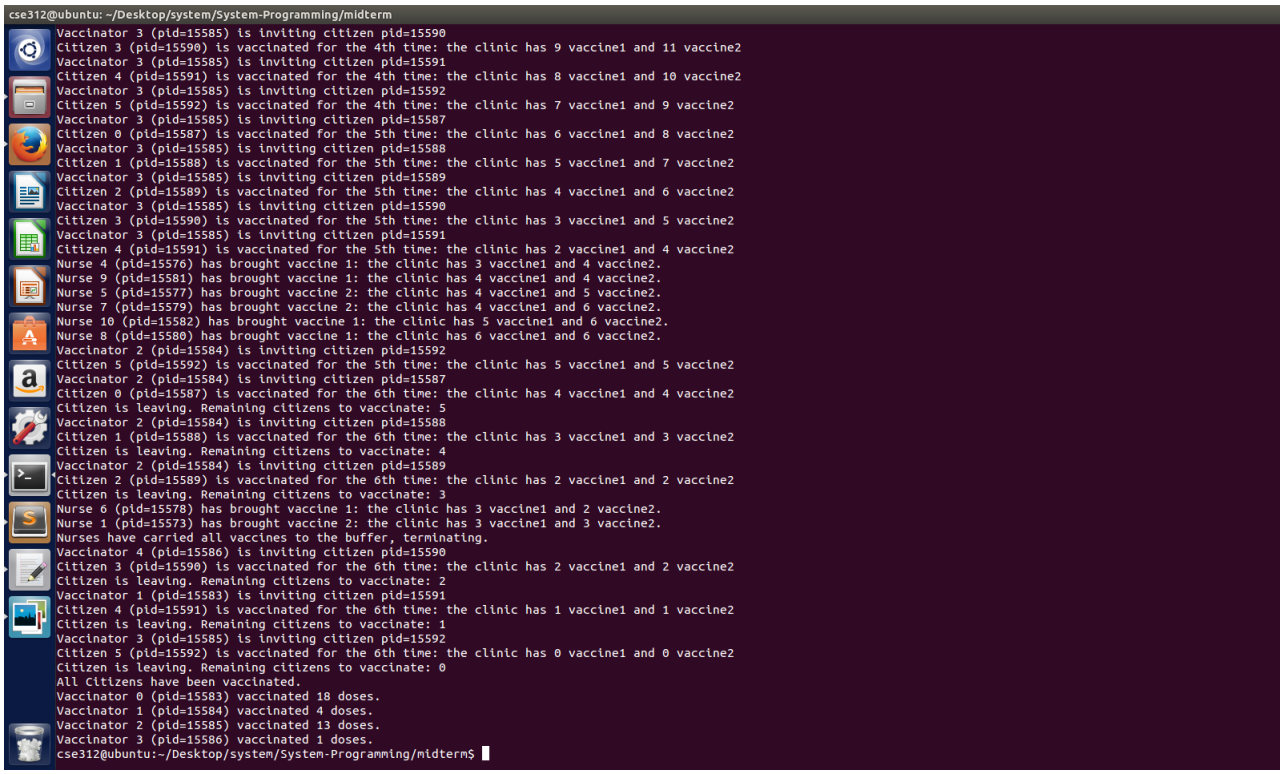


Figure 5: Running end

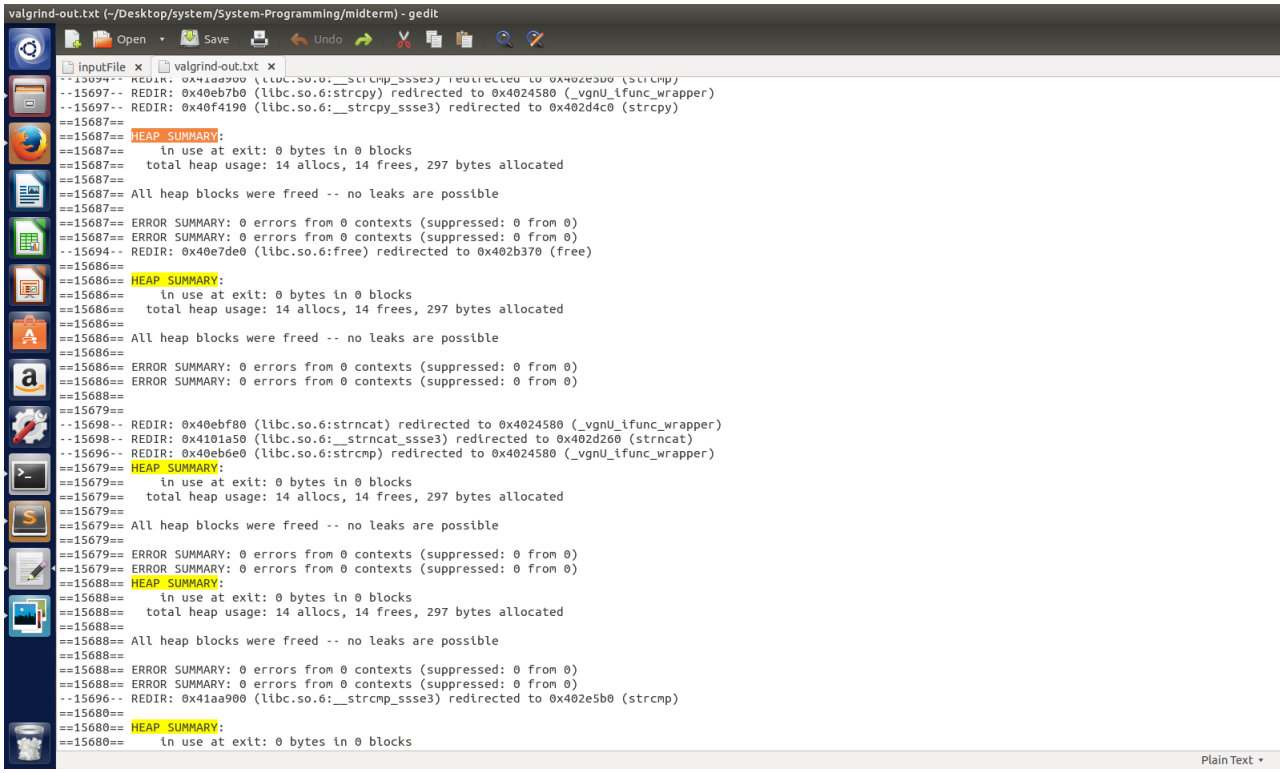


Figure 6: Valgrind output memory leak