SYSTEMS PROGRAMMING

#midterm_project

Jin Hussein

151044079

parent process :

the parent process is the process which initialize the shared memory object , initialize the counters , semaphores and mutexes . Then it forking all other processes and wait for them to finish then destroying the semaphores and mutexes and unlinks the shared memory object then terminates .

the shared memory structure :

```
struct shm{
   int vac1 ; // counter to hold the number of vac1 in the clinic
   int vac2 ; // counter to hold the number of vac2 in the clinic
   int total_vac1; //counter to hold all the vac1 in the clinic
   int total_vac2; //counter to hold all the vac2 in the clinic
   sem_t Kempty ; // semaphore to keep tracking empty spaces in the buffer/clinic , initial value = b
   sem_t Kmutex ; // binary semaphore to lock and unlock the vaccines counter vac1 and vac2 , initial value = 1
   int vaccinator_loop_counter ;//counter to hold how many times the vaccinators should  work(t*c in total )
   sem_t N_V_mutex;//mutex between the nurses and the vaccinators , when there is at least on dose the nurse should inform the vaccinator
   sem_t Qmutex; // mutex to inform the vaccinators that there are citizens waiting in the queue
   struct Queue Citizines_queue;//a queue to hold the citizens
   int detector; // a counter used to compute if a new dose is ready
   int* vaccinators_;//int array to hold the pid of the vaccinators to print it later
   int* how_many_times_vaccinated;// int array to hold the number of doses that the vaccinators vaccinated
   int vaccinators_size;  //the size of the array
};
```

The nurses process :

First it checks if there are enough space in the buffer then it loocks the critical region where it will read the input file . it reads the file if there are no more characters it terminates that process or if there are then it reads it then checks if it is one of the wanted characters then it increase the counters .

After that it checks if there is a new dose if so it notifies the vaccinator processes that it can start working .

Vaccinator process:

First it waits for the nurses to make sure that there is a dose

[sem_wait(&myshared->N_V_mutex);] ,

Then it locks the critical region where it will read and write to the shared memory .

It first checks if the vaccinators have iterated for t*c times , if so it posts the critical regions mutex (other process will be waiting there) and it terminates .

If not it takes one dose then it posts buffers semaphore for twice as it took a dose (‘1’ and ‘2’) and posts the critical regions mutex . after that it checks if there are citizens waiting in the queue

If there are it locks the critical region again then invite that citizen and vaccinate him .

Then it checks the counter again if it has reached the limit(t*c) if so it posts the mutex that comes from the nurses process as the nurses will not post it again because there will be no remaining doses

Then it posts the critical region mutex .

Citizen process :

For t times for each process it locks the critical region and adds that citizens id to the queue then unlocks the critical region then it posts the mutex to notify the vaccinators that there are citizens waiting.

Helper functions:

int found(int item , int* arr , int size) : checks if the given item is presented in the given array or not return -1 if it is not , returns the index if it is in the array .

struct Queue createQueue(int capacity , int c) . a function to create the queue with the given capacity

int dequeue(struct Queue* queue): returns the first element in the array

void enqueue(struct Queue* queue, int item) : adds the given element to the end of the array

isfilevalide(int fd,int tc2): checks if the input file contains exactly 2*t*c items of exactly t*c doses