UNIVERSITY OF THE SOUTHERN CARIBBEAN
MARACAS ROYAL ROAD, MARACAS, ST. JOSEPH.

Topic:

Essay
Assignment 3
CPTR282: Operating Systems

Instructor:

Connell Hunte

By:
Karl-Johan Bailey and Jamaal Feburary

Date: 02/03/2021

Approval…….

Aim

Based on your understanding, the text and the course material provided, create simulations of the Operating System services listed below.

Your program must be written in C++ and you are to use Data Structures as you see fit. Simplified code (Intro programming level) not using ADTs as appropriate will attract an automatic 30% reduction. Make use of stacks, trees, graphs, linked list, classes etc as you see fit. A simple program for this project is estimated to be no less than 500 lines of code.

One (1) example of a CPU scheduling algorithm (for example SJF or RR).

A simulation of process synchronization.

A simulation of a Storage Management scheme

What is CPU Scheduling?

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

CPU Scheduling

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold due to the unavailability of any resource, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient.

Priority Based Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory and time requirements

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time. Preemptive scheduling, however, is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.
- First Come First Serve Scheduling Algorithm

Applicant Class: this class stores all information on each person applying for the vaccine that is mainly based on their age. If their occupation is as a front line worker they will be given higher or lower priority.

```
class Applicant
{
public:

    Applicant();
    Applicant(string firstname, string lastname, int age, bool is_frontlineworker);

    int getAge() {
        return Applicant::age;
    }

    bool getIsFrontLineWorker() {
        return Applicant::is_frontlineworker;
    }

    string getName() {
        return first_name + " " + last_name;
    }

    bool getIsVisited() {
        return Applicant::isVisited;
    }

    void setIsVisited(bool visited) {
        Applicant::isVisited = visited;
    }

private:

    int age;
    string first_name;
    string last_name;
    bool is_frontlineworker, isVisited = false;
```

This snippet of code shows that after information of the applicant is collected it goes into an array of applicants then into a method that organizes them into a queue, which gives high priority individuals first preference.

```
    cin.ignore();

    system("cls");

}//While

//Organize Priority Queue

queue<Applicant> queue = organizePriorityQueue(applicants); //Returns the queue

cout << "List of Applicants to get COVID VACCINE from higest priority to lowest.\n" << endl;

int num = 1;

ColorSelect(14);

while (!queue.empty()) {
    Applicant applicant = queue.front();
    queue.pop();
    cout << num << ") " << applicant.getName() << " | Age: " << applicant.getAge() <<  " | Front Line Worker: " << DisplayBoolean(applicant.getIsFrontLineWorker()) << "\n" << endl;
    num++;
}
```

This part of the code entails checks to push the applicants into the queue. This method returns a queue, which would be used to display the priority list of users.

```
queue<Applicant> OS_Methods::organizePriorityQueue(Applicant* applicants) {

    int i = 0;

    queue<Applicant> queue;

    const int MAX_AGE = 75; //1st
    const int MID_AGE = 65; //2nd
    const int MIN_AGE = 16; //3rd

    while (i < APPLICANT_NUM) {

        if (applicants[i].getIsFrontLineWorker() && !applicants[i].getIsVisited()) {
            //Queue First
            applicants[i].setIsVisited(true);
            queue.push(applicants[i]);
        }
        i++;
    }

    i = 0;

    while (i < APPLICANT_NUM) {

        if (applicants[i].getAge() >= MAX_AGE && !applicants[i].getIsVisited()) {
            //Queue Second
            applicants[i].setIsVisited(true);
            queue.push(applicants[i]);
        }
        i++;
    }

    i = 0;

    while (i < APPLICANT_NUM) {

        if (applicants[i].getAge() < MAX_AGE && applicants[i].getAge() >= MID_AGE && !applicants[i].getIsVisited()) {
            //Queue Third
            applicants[i].setIsVisited(true);
            queue.push(applicants[i]);
        }
        i++;
    }

    i = 0;

    while (i < APPLICANT_NUM) {
```

These are the main parts of the mode for that part.

Process Synchronization

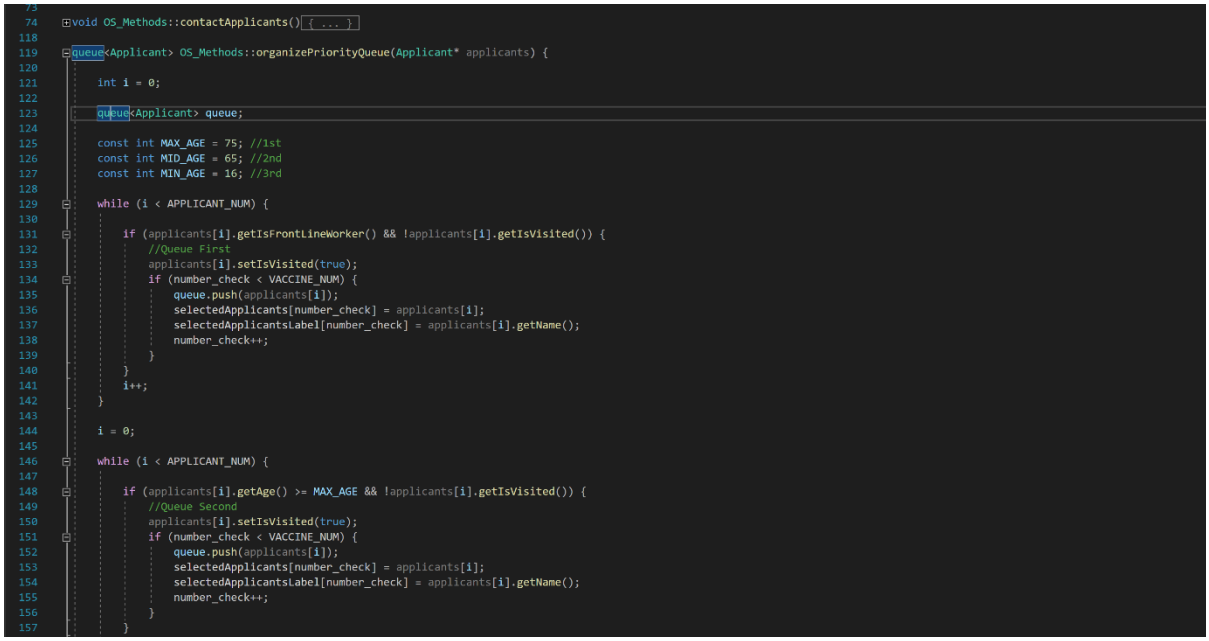
Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

This can lead to the inconsistency of shared data. Thus, the change made by one process is not necessarily reflected when other processes accesses the same shared data. To avoid this type of inconsistency of data, the processes need to be synchronized with each other.

In the code what happens is the user picks which station he wants to work from, either #1 or #2, he then can contact each applicant selected for the vaccine. When the user selects an applicant to contact from a station, that station has a lock on the applicant. No other applicant can interfere with that applicant while the contact is in progress. If one tries to access the same

applicant from another station it wouldn't allow you. This is the simulate a mutexlock on data when multiple processes are simultaneously



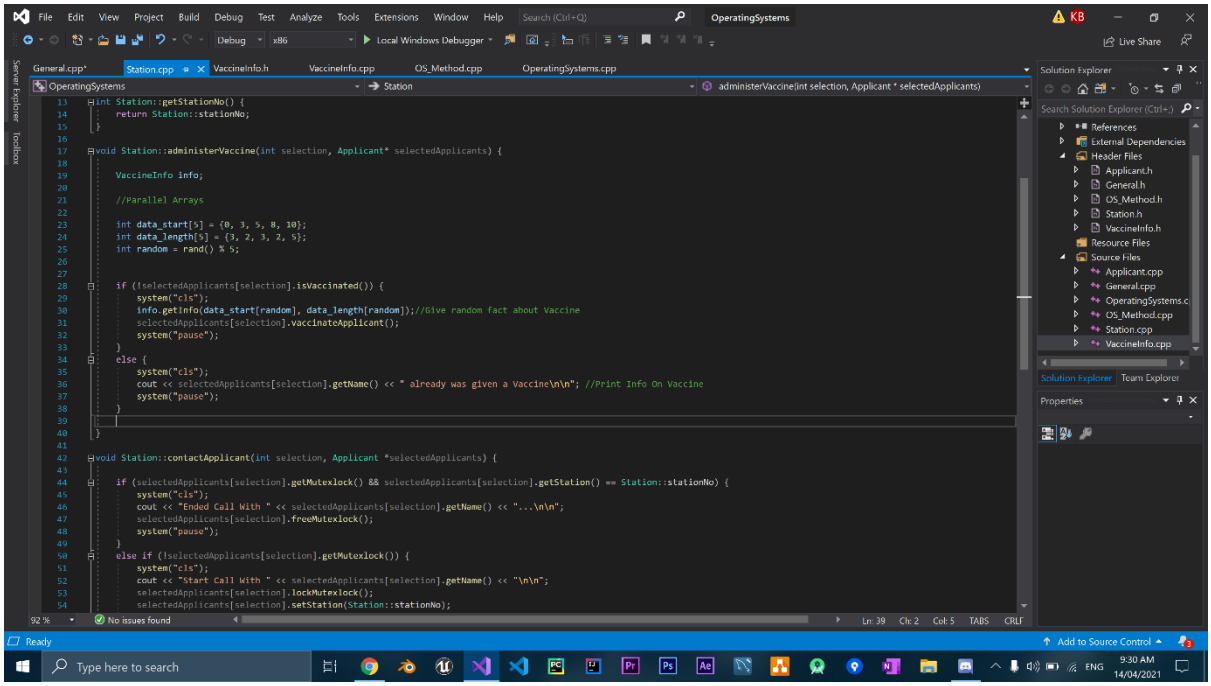
```
74  void OS_Methods::contactApplicants() { ... }
118
119 queue<Applicant> OS_Methods::organizePriorityQueue(Applicant* applicants) {
120
121     int i = 0;
122
123     queue<Applicant> queue;
124
125     const int MAX_AGE = 75; //1st
126     const int MID_AGE = 65; //2nd
127     const int MIN_AGE = 16; //3rd
128
129     while (i < APPLICANT_NUM) {
130
131         if (applicants[i].getIsFrontLineWorker() && !applicants[i].getIsVisited()) {
132             //Queue First
133             applicants[i].setIsVisited(true);
134             if (number_check < VACCINE_NUM) {
135                 queue.push(applicants[i]);
136                 selectedApplicants[number_check] = applicants[i];
137                 selectedApplicantsLabel[number_check] = applicants[i].getName();
138                 number_check++;
139             }
140         }
141         i++;
142     }
143
144     i = 0;
145
146     while (i < APPLICANT_NUM) {
147
148         if (applicants[i].getAge() >= MAX_AGE && !applicants[i].getIsVisited()) {
149             //Queue Second
150             applicants[i].setIsVisited(true);
151             if (number_check < VACCINE_NUM) {
152                 queue.push(applicants[i]);
153                 selectedApplicants[number_check] = applicants[i];
154                 selectedApplicantsLabel[number_check] = applicants[i].getName();
155                 number_check++;
156             }
157         }
```

Storage Management

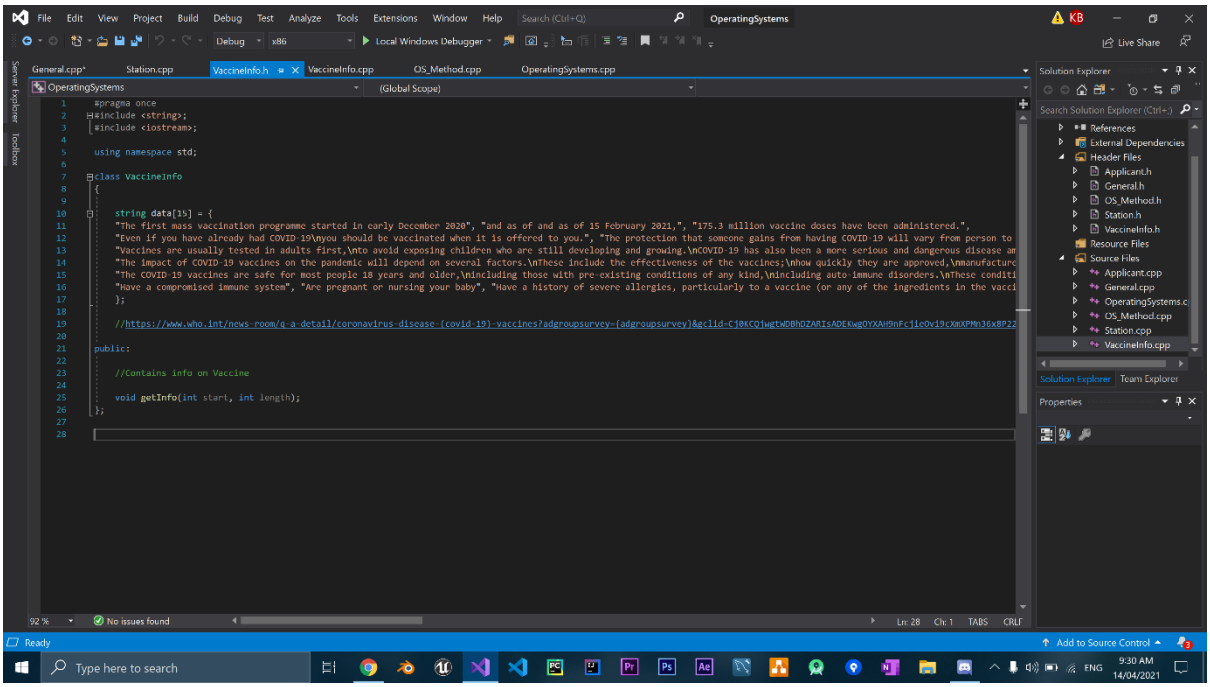
For storage allocation the contiguous storage allocation Contiguous memory allocation was used. It is a memory allocation method that allocates a single contiguous section of memory to a process or a file. This method takes into account the size of the file or a process and also estimates the maximum size, up to what the file or process can grow.

Taking into account the future growth of the file and its request for memory, the operating system allocates sufficient contiguous memory blocks to that file. Considering this future expansion and the file's request for memory, the operating system will allocate those many contiguous blocks of memory to that file.

This stores facts about the vaccines when giving them to the patients. All the information is stored in an array piece by piece so to access a particular fact about the Covid-19 vaccine you would need to know where in the index that information starts and for what length. In the code I used a randomizer to access random facts about Covid-19. This randomizer uses a parallel array to get the starting index and the length of each piece of information.



```
int Station::getStationNo() {
    return Station::stationNo;
}

void Station::administerVaccine(int selection, Applicant* selectedApplicants) {

    VaccineInfo info;

    //Parallel Arrays

    int data_start[5] = {0, 3, 5, 8, 10};
    int data_length[5] = {3, 2, 3, 2, 5};
    int random = rand() % 5;


    if (!selectedApplicants[selection].isVaccinated()) {
        system("cls");
        info.getInfo(data_start[random], data_length[random]);//Give random fact about Vaccine
        selectedApplicants[selection].vaccinateApplicant();
        system("pause");
    }
    else {
        system("cls");
        cout << selectedApplicants[selection].getName() << " already was given a Vaccine\n\n"; //Print Info On Vaccine
        system("pause");
    }
}

void Station::contactApplicant(int selection, Applicant *selectedApplicants) {

    if (selectedApplicants[selection].getMutexlock() && selectedApplicants[selection].getStation() == Station::stationNo) {
        system("cls");
        cout << "Ended Call With " << selectedApplicants[selection].getName() << "...\n\n";
        selectedApplicants[selection].freeMutexlock();
        system("pause");
    }
    else if (!selectedApplicants[selection].getMutexlock()) {
        system("cls");
        cout << "Start Call With " << selectedApplicants[selection].getName() << "\n\n";
        selectedApplicants[selection].lockMutexlock();
        selectedApplicants[selection].setStation(Station::stationNo);
```

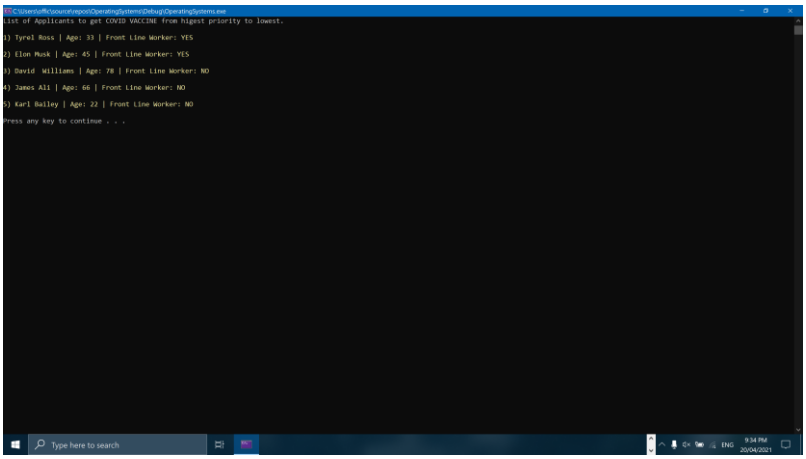


```
#pragma once
#include <string>
#include <iostream>

using namespace std;

class VaccineInfo
{

    string data[15] = {
    "The first mass vaccination programme started in early December 2020", "and as of and as of 15 February 2021,", "175.3 million vaccine doses have been administered.",
    "Even if you have already had COVID-19\nyou should be vaccinated when it is offered to you.", "The protection that someone gains from having COVID-19 will vary from person to
    "Vaccines are usually tested in adults first,\nto avoid exposing children who are still developing and growing.\nCOVID-19 has also been a more serious and dangerous disease am
    "The impact of COVID-19 vaccines on the pandemic will depend on several factors.\nThese include the effectiveness of the vaccines;\nhow quickly they are approved,\nmanufacturc
    "The COVID-19 vaccines are safe for most people 18 years and older,\nincluding those with pre-existing conditions of any kind,\nincluding auto-immune disorders.\nThese conditi
    "Have a compromised immune system", "Are pregnant or nursing your baby", "Have a history of severe allergies, particularly to a vaccine (or any of the ingredients in the vacci
    };

    //https://www.who.int/news-room/q-a-detail/coronavirus-disease-(covid-19)-vaccines?adgroupsurvey={adgroupsurvey}&gclid=Cj0KCQjwgtWDBhDZARIsADEKwgOYXAH9nFcjleOvi9cXmXPMn36x8PZZ

public:

    //Contains info on Vaccine

    void getInfo(int start, int length);
};
```

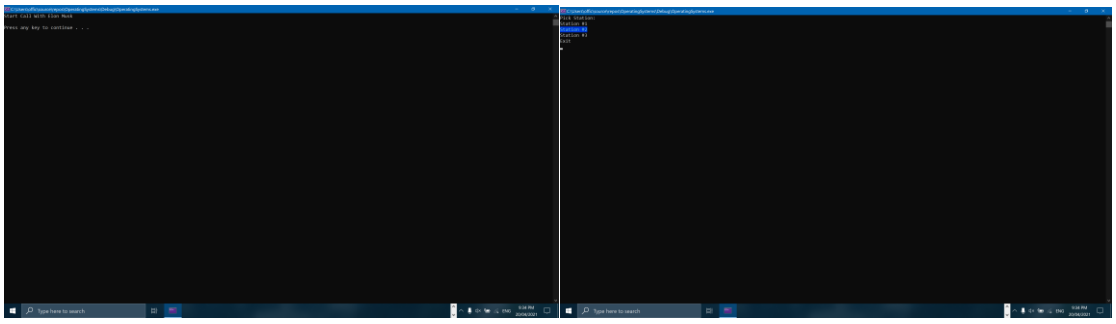
Software Instructions

The software's features are linked to each other. Each part feeds into the next.

Start by adding Applicants. These applicants are sorted in a priority queue with a maximum of 5. That means that they are only 5 COVID-19 vaccines available.



After that head on to contacting the applicants. This part will show the process synchronization in action. Once on station contacts a Applicant the other station cannot.



Afterwards you can administer the vaccines to each applicant. Tips about the vaccine will randomly appear to tell the patient.

References

- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. (2012). Operating System Concepts.9th ed.

- GeeksforGeeks, C 2018, File Allocation Methods, GeeksforGeeks, viewed 02/04/2021, https://www.geeksforgeeks.org/file-allocation-methods/

- StudyTonight, 2021, Process Synchronization, viewed 04/04/2021, https://www.studytonight.com/operating-system/process-synchronization

- WorldHealthOrganization, 2020, Coronavirus Disease (COVID): Vaccines, viewed 11/04/2021, https://www.who.int/news-room/q-a-detail/coronavirus-disease-(covid-19)-vaccines?adgroupsurvey={adgroupsurvey}&gclid=Cj0KCQjwgtWDBhDZARIsADEKwgOYXAH9nFcjieOvi9cXmXPMn36x8P22sJlPxM9ic68vi395pusO1oUaAkcfEALw_wcB