Actividad 4: Colas (fundamentos)

EDyA 361075 Carlos Eduardo Sánchez Torres

6 de marzo de 2021

1. Descripción de la actividad

Implementar un sistema de colas para brindar la atención en un módulo de atención, en el cual se pueden abrir más de un módulo dependiendo de la cantidad de clientes en espera.

Defina el giro del negocio, las políticas de cada módulo de atención al cliente (módulos VIP, para adultos mayores, para discapacitados, para cliente frecuente, trámites rápidos o lentos, etc.)

Defina cuántos módulos existirán y los tiempos de atención

Simule la llegada, atención y salida del cliente utilizando generadores pseudoaleatorios.

2. Resultados

La estructura de datos, cola, el primero en entrar el último en salir, cuyos elemento principal son cabeza y la parte trasera. Para encolar, insertar un elemento se lo agrega a la parte trasera y para desencolar sacamos la cabeza. El pseudocódigo asume que $n = Q.length$.

```
ENQUEUE(Q,x)
Q[Q.tail] = x
if Q.tail == Q.length
    Q.tail = 1
else Q.tail = Q.tail + 1
```

Listing 1: Algoritmo para encolar. $\mathcal{O}(1)$

```
DEQUEUE(Q,x)
Q[Q.tail] = x
if Q.tail == Q.length
    Q.tail = 1
else Q.tail = Q:tail + 1
```

Listing 2: Algoritmo para desencolar. $\mathcal{O}(1)$

3. Anexos

Simulación no realista de la distribución de una vacuna genérica COVID-19, consideran 4 módulos: urgente-importante, no urgente-importante, urgente-no importante y urgente-importante. Con tiempos de atención aleatorios y con 4 hilos generadores. Para ver el resto del código: [2].

```
void Queue<T>::enqueue(T element) {
    this->tail = new QueueNode<T>(element, this->tail);
    this->createHead();
    this->increase();
}
```

Figura 1: Encolar.

```
T Queue<T>::dequeue() {
    this->ensureIsEmpty();
    this->decrease();
    return this->head->moveToPrevious();
}
```

Figura 2: Desencolar.

```
void simulate() {
    PatientPublisher patientPublisher(patientPrioritizer:new PatientPrioritizer());
    cout << "Bootstrapping...\n";
    thread urgentImportantPeopleFirstThread (notifyAboutUrgentImportantPeopleThread,patientPublisher);
    thread urgentImportantPeopleSecondThread (notifyAboutUrgentImportantPeopleThread,patientPublisher);
    thread noUrgentImportantPeopleThread (notifyNoUrgentImportantPeopleThread,patientPublisher);
    thread urgentNoImportantPeopleThread (notifyUrgentNoImportantPeopleThread,patientPublisher);
    thread otherPeopleThread (notifyOtherPeopleThread,patientPublisher);
    urgentImportantPeopleFirstThread.join();
    urgentImportantPeopleSecondThread.join();
    noUrgentImportantPeopleThread.join();
    urgentNoImportantPeopleThread.join();
    otherPeopleThread.join();
    cout << "Completed.\n";
}
```

Figura 3: Simular.

```
void prioritize(Patient* patient) {
    Vaccines::GetInstance()->increase();
    switch(patient->getPriority()) {
        case URGENT_IMPORTANT:
            this->urgentImportantPatientQueue->attend(patient);
            break;
        case NO_URGENT_IMPORTANT:
            this->noUrgentImportantPatientQueue->attend(patient);
            break;
        case URGENT_NO_IMPORTANT:
            this->urgentNoImportantPatientQueue->attend(patient);
            break;
        case NO_URGENT_NO_IMPORTANT:
            this->noUrgentNoImportantPatientQueue->attend(patient);
            break;
    }
}
```

Figura 4: Principal.

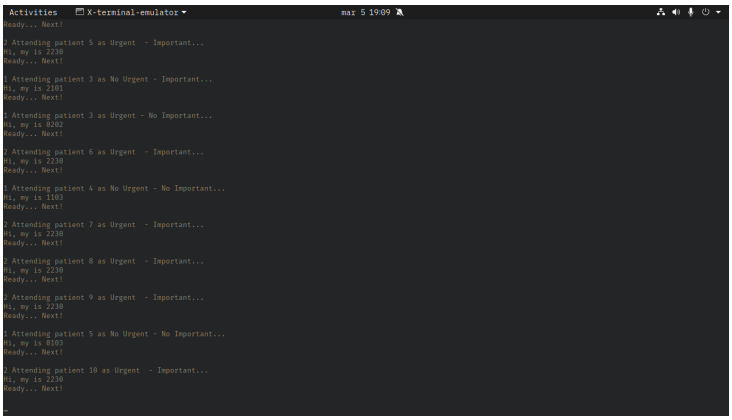

Figura 5: Resultados.

Referencias

[1] Cormen, T. H., &; Leiserson, C. E. (2009). Introduction to Algorithms, 3rd edition. In Introduction to algorithms, 3rd edition.

[2] sanchezcarlosjr. 2021. sanchezcarlosjr/uabc. GitHub. Retrieved March 6, 2021 from https://github.com/sanchezcarlosjr/uabc/tree/master/src/introduction-to-algorithms/week4-practice1