Jacob Hreshchyshyn

Rithvik Arun

Analysis:

Problem Definition:

The task is to design a set of live algorithms that interact with one another to manage an interlocked door. This interlocked door system is composed of two doors that lead into an inner room. Only one door may be opened at a time even during emergencies. Keys are used to open the doors. The same key can be used on either door (assumption). In order to access the inner room from one of the access corridors, one only requires a key. In order to exit the inner room through one of the doors, both the key has to be used on the door and the control room has to approve access. An open door must wait until signaled to close. A timer is used to signal an open door to close after the timer ends (assumption). The control room can manually delay the timer to allow for more people to enter through that open door (partial assumption). In order to exit the inner room, the control room must signal the door to open.

In the case of two people attempting to access both doors from the access corridors, the process of choosing which door opens is not deterministic, but once one door is signaled to open and opens, the other door cannot open until the first door closes.

The available process synchronization tools to choose from for our algorithm, as defined in K, are mutexes and semaphores.

Mutexes are mechanisms that can be either locked or unlocked allowing for management of one specific resource.

Semaphores are similar to mutexes in that they act as locking and unlocking mechanisms. The difference is that they allow multiple resources to be handled.

Assumptions:

- The same key can be used on either door. Both door A and door B have the same key and they can be interchanged to open one another.
- A timer is used to signal an open door to close. There is a timer used to keep track of how long the door was open. If the time that the door is open exceeds the limit of the timer, it will close unless opened again or the control room manually delays the timer.
- The control room is able to delay the timer to allow for more time for people to enter or exit the passage room.

Metrics:

- The system should have only one resource that doors use to open in order to improve security. This helps satisfy the mutual exclusion criterion for this particular critical section problem.
- The time it takes for a door to open and close after a key is entered from the access corridor or after the control room signals the door to open should be 60 seconds, ensuring a deterministic waiting time that helps satisfy this critical section problem. This is not to say that the waiting time will be bounded, since this system is not expected to terminate.

Design/Choice:

The best solution would be to use a mutex-based algorithm for this system. Below is a simple pseudocode example of the workings of the system. This pseudocode example shows the process of attempting to open a door from one of the access corridors.

```
pthread_mutex_t doorLock;

void * interlockedDoorOperation(void * doorProcess){
    pthread_mutex_lock(&doorLock); // represents key used to open door

    create thread to handle timer //represents a timer for the door (runs on a seperate thread)
        create new timer object //create timer object
        start timer //start the timer

        while true
            if timer.currentTime > timer.limit //checks if timer limit is exceeded
                if control room dose not allow more time // if timer limit is exceeded we check with the control
                                                         //room to see if
                                                         //they can manually delay the timer
                pthread_mutex_unlock(&doorLock); // both checks return false close the door and destroy the timer
                stop timer
                destory timer

    //let person walk through

    pthread_mutex_unlock(&doorLock); // represents closed door
}

void main(){
    pthread_mutex_init(&doorLock, NULL); //initialize mutex

    //feed threads into this (2 threads for this system, A & B, one for each door),
    //threads can come in at any time
    //this is a representation of door A requesting to open before door B
    interlockedDoorOperation(doorProcessA)
    interlockedDoorOperation(doorProcessB)

    pthread_mutex_destroy(&doorLock);//destroy mutex
}
```

This algorithm allows multiple processes (threads) in the runner function but only allows one to execute at a time. These processes represent the doors in our system. When a door wants to be opened, it goes through the algorithm and first checks to see if another door is open (pthread_mutex_lock(&doorLock). If there is another door opened the system will wait until that door is closed since the mutex has to be unlocked in order for a door to open. This makes sure

that only one door can be opened at a time. Once the door is open we have a separate thread that handles the timer as well as the control room input. This thread constantly checks to see if the door has been opened longer than the timer allows. If it has, then it checks with the control to see if the timer can be delayed to allow for more time for the door to be opened. If the time limit is exceeded and the control room does not allow for any more time for the door to stay open, then the door will be closed (pthread_mutex_unlock(&doorLock) and another person is able to open one of the doors.

There would also be another mutex-based algorithm that handles an individual opening a door from the inner room. Instead of a mutex existing to represent the resource needed for a door to open, the resource would represent a notification to the control room. This resource would then only be released once the control room opens the door.

Justification:

A mutex-based algorithm implementation is the best design for this system mainly because of the fact that we only have one resource running at a time. Since only one door can be opened at a time we can safely say that we only have two options, either the door is open or closed. We can represent this using the lock and unlock mechanisms that a mutex provides.

A semaphore implementation would have the similar function of locking the resources needed, in this case, for a door to open. Semaphores, however, have the potential to have multiple resources available for multiple processes. This means that multiple doors can be opened at the same time if multiple doors have access to several resources that allow the doors to open. However, because the problem description specifies that only one door may be opened at a time, there is no need for a semaphore implementation to potentially extend the system to allow multiple doors to be open at the same time.

Analysis:

<u>Problem Definition:</u>

The task is to identify the most important scheduling criteria for a mobile device operating system that is responsible for employee identification and access control. It will also be responsible for reporting local sensor readings to a main server. These sensor readings will be run when the mobile device is within 10 feet of another inactive permanent sensor (assumption). Such a sensor will be found through a passive process that utilizes less battery life than the main sensor reading process (assumption). Readings can also be run remotely, though no readings can be run when the mobile device's battery life is below 30% (assumption). Since the process of running sensor readings may drain battery life quickly, the report of sensor readings should be sent to the server within 60 seconds of receiving readings (assumption). To clarify, these readings are measuring local atmospheric carbon dioxide concentrations using infrared sensors (assumption).

The available criteria to choose from, as defined in K, are CPU Utilization, Throughput, Turnaround Time, Waiting Time, and Response Time.

CPU Utilization is how much of the CPU is actually used of the possible load that could be put on the CPU.

Throughput is the number of tasks completed after some period of time.

Turnaround Time is the total completion time for a task.

Waiting Time is the time a task spends in the ready queue.

Response Time is the time spent until a task first gives a result.

Assumptions:

- Mobile device readings may be activated when the device detects another permanent sensor from 10 feet away.
- A permanent sensor detection process that does not drain battery life as quickly as the main reading process should be used to detect permanent sensors.
- Readings will not be run when the battery life is below 30%.
- Sensor readings are measuring local atmospheric carbon dioxide concentrations.

Metrics:

- These infrared sensors should be capable of absorbing gas wave samples of wavelengths between 4μm - 5μm.
- The mobile device must be able to detect permanent sensor signals from 10 feet away.
- The mobile device must be able to report a batch of readings within 60 seconds of detecting readings.

Design/Choice:

Given the above problem description and metrics, the most important scheduling criterion is Throughput.

Justification:

Throughput is the most important scheduling criterion for this system largely because of the need to conserve the battery life of the mobile device. If it takes too long for the system to detect readings and send those readings to the server, the mobile device will not be able to send as much data to the server since the battery life will continue to drain as long as the reading process is active. This means that the system must prioritize the rapid processing of batches of readings so that as much reading data is sent to the server before the battery life of the mobile device falls below 30%. Additionally, the number of batches a server receives within a time

frame, indicating how many batch forming processes were completed, can be easily measured, meaning that Throughput would be the best guide in determining the efficiency of reading processing.

While Response Time is also an important criterion, because it is concerned with the time spent for a single task to give a result, it would not be as good a measure for task completion within a timeframe as Throughput, which is concerned with a collection of tasks completed within a timeframe.

Turnaround Time and Waiting Time are also useful measures of the efficiency of task completion. However, because the batches of readings are being sent in discrete sets, small improvements in Turnaround Time and Waiting Time will likely not affect the number of batches sent to the server. If the focus is made on maximizing Throughput, Turnaround Time and Waiting Time will also be improved.

Lastly, CPU Utilization is another important scheduling criterion since it would help in determining the correct balance between obtaining readings from the environment and processing those readings in batches to be sent to the server. However, this does not tell us directly how quickly that data is sent to the server. While effectively balancing between data collection and CPU processing will increase the number of tasks completed, focusing on Throughput helps provide a more visible performance target. Once again, focusing on improving Throughput will lead to efforts in improving CPU Utilization.