OPERATING SYSTEM

DESIGN EXPERIMENT



**CIGARETTE SMOKERS’ PROBLEM**

Submitted by:

|  |  |
| --- | --- |
| HAFIZ AHMAD | 19L-1316 |

Submitted to:

HAMZA YOUSAF

Kamran Jamal

MAY 31, 2022

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Lahore

## Introduction

1. **Problem Analysis**

## Design Requirements

1. **Feasibility Analysis**

## Possible Solutions

1. **Preliminary Design**

## Design Description

## Software Simulation

## Experimental Results

1. **Performance Analysis**
2. **Future Scope**

## Social and Cultural Implications

1. **Conclusion References**

*In software engineering, synchronization alludes to one of two unmistakable yet related ideas: synchronization of cycles, and synchronization of information. Process synchronization alludes to the possibility that different cycles are to sign up or handshake at one point, to agree or focus on a specific grouping of activity. Information synchronization alludes to keeping numerous duplicates of a dataset in cognizance with each other, or to keep up with information honesty. Process synchronization natives are usually used to carry out information synchronization. The requirement for synchronization doesn't emerge just in multi-processor frameworks yet for any sort of simultaneous cycles; even in single processor frameworks. Referenced beneath are a portion of the fundamental requirements for synchronization:Forks and Joins: When a task shows up at a fork point, it is parted into N sub-occupations which are then overhauled by n errands. In the wake of being adjusted, each sub-work holds on until any remaining sub-positions are finished handling. Then, at that point, they are joined once more and leave the framework. Hence, equal programming requires synchronization as every one of the equal cycles hang tight for a few different cycles to occur.Producer-Consumer: In a maker customer relationship, the buyer cycle is reliant upon the maker cycle till the essential information has been created. Whenever different cycles are reliant upon an asset and they need to get to it simultaneously, the working framework needs to guarantee that only one processor gets to it at a given moment. This decreases simultaneousness. String synchronization is characterized as a component which guarantees that at least two simultaneous cycles or strings don't all the while execute some specific program fragment known as basic segment. Cycles' admittance to basic segment is constrained by utilizing synchronization strategies. At the point when one string begins executing the basic area (serialized section of the program) the other string ought to hold on until the main string wraps up. In the event that legitimate synchronization techniques[1] are not applied, it might cause a race condition where the upsides of factors might be unusual and change contingent upon the timings of setting switches of the cycles or threads.For model, assume that there are three cycles, in particular 1, 2, and 3. Each of them three are simultaneously executing, and they need to share a typical asset (basic segment) as displayed in Figure 1. Synchronization ought to be utilized here to keep away from any struggles for getting to this common asset. Subsequently, when Process 1 and 2 both attempt to get to that asset, it ought to be doled out to just a single cycle at a time. Assuming it is appointed to Process 1, the other interaction (Process 2) necessities to hold on until Process 1 liberates that resourceOther than shared rejection, synchronization likewise manages the accompanying:*

*• halt, which happens when many cycles are sitting tight for a common asset (basic segment) which is being held by another interaction. For this situation, the cycles simply continue to stand by and execute no further;*

*• starvation, which happens when an interaction is standing by to enter the basic segment, however different cycles hoard the basic segment, and the primary cycle is compelled to endlessly pause;*

*• need reversal, which happens when a high-need process is in the basic segment, and it is intruded on by a medium-need process. This infringement of need rules can occur in specific situations and may prompt serious outcomes progressively frameworks;*

*• in the middle of stalling, which happens when a cycle habitually surveys to decide whether it approaches a basic segment. This regular surveying burglarizes handling time from different cycles.*

There are four cycles in this issue: three smoker processes and a specialist interaction. Each of

the smoker cycles will make a cigarette and smoke it. To make a cigarette requires tobacco,

paper, and matches. Every smoker cycle has limitless stockpile of one of the three things. i.e., one process has tobacco, another has paper, and a third has matches. The specialist has a boundless stockpile of each of the three. The specialist puts two of the three things on the table, and the smoker that has the third thing makes the cigarette. Synchronize the cycles.

As far as information the specialist arbitrarily puts two of the three fixings on the table indiscriminately with no decent request. The specialist isn't allowed to have a lot of insight into the asset needs of smokers; i.e., the specialist can't wakeup a smoker straightforwardly. Each time the specialist makes two assets accessible, it should look out for a condition variable for a smoker to smoke before it can make any extra assets accessible. The specialist is simply permitted to convey by flagging the accessibility of an asset utilizing a condition variable.

After not set in stone by the program which smoker is smoking a result is shown on the screen showing what assets are given by the specialist and which smoker is actuated. Moreover, some kind of sign is likewise shown that the smoker has completed the process of smoking.

[*Note: Discuss about the feasibility of the project in terms of Time Management and Cost Management in particular. While doing a project there are multiple resources under use and there are constraints while using the resources and hence you need good management to complete project subject to all the constraints. Feasibility study is done to make sure whether it is possible to meet the goal or not. Remove this content while writing your report.*]

Cost management:

No cost used in doing this project.

Time management:

|  |  |
| --- | --- |
| Research | 3 days |
| Coming up with Solutions | 1 day |
| Finalizing the solution | 2 days |
| Implementing on C | 3 days |
| Finalizing the project | 1 day |

There are four cycles in this issue: three smoker processes and a specialist interaction. Every one of the smoker cycles will make a cigarette and smoke it. To make a cigarette requires tobacco, paper, and matches. Every smoker cycle has one of the three things. I.e., one cycle has tobacco, another has paper, and a third has matches. The specialist has a boundless stock of each of the three. The specialist puts two of the three things on the table, and the smoker that has the third thing makes the cigarette. Synchronize the cycles. This appears to be a genuinely simple arrangement. The three smoker cycles will make a cigarette and smoke it. On the off chance that they can't make a cigarette, then, at that point, they will nod off. The specialist interaction will put two things on the table, and wake up the suitable smoker, and afterward nod off. All semaphores aside from lock are instated to 0. lock is introduced to 1, and is a mutex variable. The smoker promptly rests. At the point when the specialist puts the two things on the table, then, at that point, the specialist will awaken the fitting smoker. The smoker will then get the things, and wake the specialist. While the smoker is smoking, the specialist can put two things on the table, and wake an alternate smoker (on the off chance that the things put aren't something similar). The specialist rests following setting the things out. This is the kind of thing like the maker buyer issue with the exception of the maker can deliver 1 thing (albeit a decision of 3 sorts of things) at a time.There are four processes in this problem: three smoker processes and an agent process. Each of the smoker processes will make a cigarette and smoke it. To make a cigarette requires tobacco,paper, and matches. Each smoker process has infinite supply of one of the three items. one process has tobacco, another has paper, and a third has matches. The agent has an infinite supplyof all three. The agent places two of the three items on the table, and the smoker that has thethird item makes the cigarette. Synchronize the processes.

The process takes on three terminals.

Table

Description automatically generated

[*Note: Discuss all the blocks of your design in detail. Remove this content while writing your report.*]

* The plan for the venture incorporates four strings three smoker strings and a specialist string. The specialist string is the driver for this entire effort. On executing the specialist string haphazardly puts the fixings on the table (two of the three). A condition is set up which figures out which smoker string is to be initiated, when enacted the other two smoker strings are impeded utilizing semaphores. At the point when the smoker string has gotten done with smoking after a postpone utilizing a defer work the specialist string again puts the fixings on the table and it goes into a boundless stop as the fixings are limitless.

Agent code:-

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/stat.h>

int main(int argc, char \*arg[])

{

int count=0,material=0;

char \*n1="/my\_semaphore1";

char \*n2="/my\_semaphore2";

char \*n3="/my\_semaphore3";

char \*n4="/my\_semaphore4";

sem\_t\* sem1; //for matchsticks

sem\_t\* sem2; //for paper

sem\_t\* sem3; //for tobacco

sem\_t\* sem4; //for missing material

int value=0;

sem1=sem\_open(n1,O\_CREAT,0666,value);

sem2=sem\_open(n2,O\_CREAT,0666,value);

sem3=sem\_open(n3,O\_CREAT,0666,value);

sem4=sem\_open(n4,O\_CREAT,0666,value);

while(1)

{

material=rand()%3;

printf("Agent has put something on the table:\n");

if(material==0)

{printf("tobacco\n");

sem\_post(sem1);

sem\_post(sem2);

}

else if(material==1)

{printf("paper\n");

sem\_post(sem1);

sem\_post(sem3);

}

else if(material==2)

{printf("matchsticks\n");

sem\_post(sem3);

sem\_post(sem2);

}

else

{

printf("Agent is not working properly");

}

sem\_wait(sem4);

count++;

if(count==10)

{break;}

}

sem\_close(sem1);

sem\_close(sem2);

sem\_close(sem3);

sem\_close(sem4);

sem\_unlink(n1);

sem\_unlink(n2);

sem\_unlink(n3);

sem\_unlink(n4);

return 0;

}

Smoker 1:-

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/stat.h>

#include <unistd.h>

int main(int argc, char \*arg[])

{

char \*n1="/my\_semaphore1";

char \*n2="/my\_semaphore2";

char \*n4="/my\_semaphore4";

sem\_t\* sem1; //for matchsticks

sem\_t\* sem2; //for paper

sem\_t\* sem4; //for missing material

int value=0;

sem1=sem\_open(n1,O\_CREAT,0666,value);

sem2=sem\_open(n2,O\_CREAT,0666,value);

sem4=sem\_open(n4,O\_CREAT,0666,value);

while(1)

{

sem\_wait(sem1);

if(sem\_trywait(sem2)==0)

{

printf("ðŸ˜—ðŸš¬\n");

sleep(2);

sem\_post(sem4);

}

else

sem\_post(sem1);

}

}

Smoker 2:-

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/stat.h>

#include <unistd.h>

int main(int argc, char \*arg[])

{

char \*n1="/my\_semaphore1";

char \*n3="/my\_semaphore3";

char \*n4="/my\_semaphore4";

sem\_t\* sem1; //for matchsticks

sem\_t\* sem3; //for tobacco

sem\_t\* sem4; //for missing material

int value=0;

sem1=sem\_open(n1,O\_CREAT,0666,value);

sem3=sem\_open(n3,O\_CREAT,0666,value);

sem4=sem\_open(n4,O\_CREAT,0666,value);

while(1)

{

sem\_wait(sem3);

if(sem\_trywait(sem1)==0)

{

printf("ðŸ˜—ðŸš¬\n");

sleep(2);

sem\_post(sem4);

}

else

sem\_post(sem3);

}

}

Smoker 3:-

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/stat.h>

#include <unistd.h>

int main(int argc, char \*arg[])

{

char \*n2="/my\_semaphore2";

char \*n3="/my\_semaphore3";

char \*n4="/my\_semaphore4";

sem\_t\* sem2; //for paper

sem\_t\* sem3; //for tobacco

sem\_t\* sem4; //for missing material

int value=0;

sem2=sem\_open(n2,O\_CREAT,0666,value);

sem3=sem\_open(n3,O\_CREAT,0666,value);

sem4=sem\_open(n4,O\_CREAT,0666,value);

while(1)

{

sem\_wait(sem2);

if(sem\_trywait(sem3)==0)

{

printf("ðŸ˜—ðŸš¬\n");

sleep(2);

sem\_post(sem4);

}

else

sem\_post(sem2);

}

}

Text

Description automatically generated

The arrangement proposed by Parnas utilizes three assistant strings called "pushers" that

answer the signs from the specialist, monitor the accessible fixings,

furthermore, signal the suitable smoker.

The extra factors and semaphores are

Smokers issue hint

The boolean factors show whether a fixing is on the table.

The pushers use tobaccoSem to flag the smoker with tobacco, and the other

semaphores moreover.

Here is the most ideal code for one of the pushers:

This pusher awakens any time there is tobacco on the table. In the event that it finds

isPaper valid, it realizes that Pusher B has proactively run, so it can flag the

smoker with matches. Essentially, on the off chance that it finds a match on the table, it can flag

the smoker with paper.

However, on the off chance that Pusher A runs first, it will find both isPaper and isMatch misleading.

It can't flag any of the smokers, so it sets isTobacco.

Different pushers are comparable. Since the pushers do all the genuine work, the

smoker code is inconsequential:

Smoker with tobacco

Parnas presents a comparable arrangement that gathers the boolean factors, bitwise, into a whole number, and afterward involves the number as a record into a variety of

semaphores. That way he can try not to utilize conditionals (one of the counterfeit

imperatives). The subsequent code is a touch more succinct, yet its capacity isn't as

self-evident.

Just for practceing and solving a problem task on synchronization so we learn and practice on

synchronization.

Its bad effect on social and cultural implications specially on childs who see this project and this like a game

and we know that smoking is not good for our health.

This appears to be a genuinely simple arrangement. The three smoker cycles will make a cigarette and smoke it. In the event that they can't make a cigarette, then they will nod off. The specialist cycle will put two things on the table, and wake up the suitable smoker, and afterward fall asleep. All semaphores aside from lock are instated to 0. lock is instated to 1, and is a mutex variable.The smoker promptly rests. At the point when the specialist puts the two things on the table, then, at that point, the specialist will awaken the proper smoker. The smoker will then, at that point, get the things, and wake the specialist. While the smoker is smoking, the specialist can put two things on the table, and wake an alternate smoker (on the off chance that the things set aren't something very similar). The specialist dozes following putting the things out. This is the sort of thing like the maker customer issue with the exception of the maker can create 1 thing (albeit a decision of 3 sorts of things) at a time

1. https://pages.mtu.edu/~shene/NSF-3/e-Book/SEMA/TM-example-smoker.html
2. https://gist.github.com/evanpurkhiser/5385112
3. http://www.edutalks.org/downloads/A%20C%20PROGRAM%20TO%20IMPLEMENT%20THE%20SMOKER%27S%20PROBLEM%20USING%20SEMAPHORES.pdf
4. https://mycareerwise.com/content/cigarette-smokers-problem/content/exam/gate/computer-science
5. https://www.cs.umd.edu/~hollings/cs412/s96/synch/smokers.html
6. https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/CigSmokers.html
7. https://en.wikipedia.org/wiki/Cigarette\_smokers\_problem
8. https://learningcomputersciencemadeeasy.wordpress.com/2017/04/08/cigarette-smoker-problem/
9. https://dl.acm.org/doi/pdf/10.1145/360680.360709

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN | Attribute | Complex Activities | Mapping of Experiment | Evaluator’s Comments |
| 1 | **Range of resources** | Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies). | The design experiments involve the use of diverse resources. Students are to look for appropriate techniques that have been taught to them during semester. Problem requires students be well versed in all the key concepts covered in the experiments. |  |
| 2 | **Level of interaction** | Require resolution of significant problems arising from interactions between wide-ranging or conflicting technical, engineering or other issues. | Design of the experiment requires students to address conflicting technical issues while designing the solution/algorithm as the solution/algorithm needs to be accurate i.e to handle all the exceptions and it should be efficient i.e good in terms of time/space complexity. |  |
| 3 | **Innovation** | Involve creative use of engineering principles and research-based knowledge in novel ways. | N/A |  |
| 4 | **Consequences to society and the environment** | Have significant consequences in a range of contexts, characterized by difficulty of prediction and mitigation. | N/A |  |
| 5 | **Familiarity** | Can extend beyond previous experiences by applying principles-based approaches | This design experiment requires the students to develop a solution which is an extension to all the key concepts delivered during the semester particularly the experiments dealing with the interprocess communication. |  |