

OPERATING SYSTEM

PROJECT REPORT

SYSTEM CALL USING

SEMAPHORE

CHAIN SMOKER PROBLEM

**TEAM MEMBERS:**

Ehtesham Zafar Jan Roll# 20k-1655

Syed Muhammed Hassan Ali Roll# 20k 1052

Syed Muhammed Raza Abidi Roll# 20k-1061

**BSE-4B**

# INTRODUCTION

This project is dedicated to creating a system call that deals with the chain smoker problem. A system call is a request for a service that is made by the application programs to the operating system; these can be either user system call (without kernel intervention) or kernel system call (with kernel intervention).

**FEATURES**

Main function deals with the creation, and deletion of threads, and semaphores. This problem has four processes, three smoker processes, and one agent process. Each of the smoker procedures will create and smoke a cigarette. Tobacco, paper, and matches are needed to produce a cigarette. One of the three components is present in each smoking procedure. To put it another way, one procedure uses tobacco, another uses paper, and yet another uses matches. All three are infinitely available to the agent. Two of the three objects are placed on the table by the agent, and the smoker with the third item lights the cigarette.

**TOOLS AND TECHNOLOGY**

Programming Language: C language

VMware Work Station 16   
Platform: Ubuntu 16.04

**CODE SNIPPETS**

#include <stdio.h>

#include <unistd.h> /\* Symbolic Constants \*/

#include <sys/types.h> /\* Primitive System Data Types \*/

#include <errno.h> /\* Errors \*/

#include <stdio.h> /\* Input/Output \*/

#include <stdlib.h> /\* General Utilities \*/

#include <pthread.h> /\* POSIX Threads \*/

#include <string.h> /\* String handling \*/

#include <semaphore.h> /\* Semaphore \*/

#include <sys/syscall.h>

#include <linux/kernel.h>

sem\_t more\_needed;

sem\_t match;

sem\_t paper;

sem\_t tobacco;

void \*agent ()

{

int i=0;

int sm=1;

int s=0,p=0,m=0;

while (1)

{

int number = rand() % 3;

if(i==10){

printf("\n\nTotal number of time smoker with ciggerete smoked: %d\n",s);

printf("\nTotal number of time smoker with paper smoked: %d\n",p);

printf("\nTotal number of time smoker with match smoked: %d\n",m);

exit(0);

}

sleep(1);

switch (number)

{

case 0: sem\_post (&match); /\* match and paper \*/

sem\_post (&paper);

syscall(333,"Agent has put match and paper on the table\n");

//printf("Agent has put match and paper on the table\n");

printf("Smoking %d time\n", sm++);

s++;

break;

case 1: sem\_post (&match); /\* match and tobacco \*/

sem\_post (&tobacco);

syscall(333,"Agent has put match and tobacco on the table\n");

//printf("Agent has put match and tobacco on the table\n");

printf("Smoking %d time\n", sm++);

p++;

break;

case 2: sem\_post (&paper); /\* tobacco and paper \*/

sem\_post (&tobacco);

syscall(333,"Agent has put paper and tobacco on the table\n");

//printf("Agent has put paper and tobacco on the table\n");

printf("Smoking %d time\n", sm++);

m++;

break;

}

sem\_wait (&more\_needed); /\* wait for request for more \*/

i++;

}

}

void \*smoker\_with\_tobacco ()

{

while (1)

{

sem\_wait (&match); /\* grab match from table \*/

if (sem\_trywait (&paper) == 0) /\* grab paper \*/

{

/\* roll cigarette and smoke \*/

syscall(333,"match and paper feched");

syscall(333,"smoker with tobacco is smoking\n");

//printf("tobacco smoking\n");

sleep(0.5);

sem\_post (&more\_needed); /\* signal to agent \*/

}

else sem\_post (&match); /\* drop the match \*/

}

}

void \*

smoker\_with\_match ()

{

while (1)

{

sem\_wait (&paper); /\* grab match from table \*/

if (sem\_trywait (&tobacco) == 0) /\* grab paper \*/

{

/\* roll cigarette and smoke \*/

syscall(333,"tobacco and paper feched");

syscall(333,"smoker with match is smoking\n");

//printf("match smoking\n");

sleep(0.5);

sem\_post (&more\_needed); /\* signal to agent \*/

}

else sem\_post (&paper); /\* drop the match \*/

}

}

void \*

smoker\_with\_paper ()

{

while (1)

{

sem\_wait (&tobacco); /\* grab match from table \*/

if (sem\_trywait (&match) == 0) /\* grab paper \*/

{

/\* roll cigarette and smoke \*/

syscall(333,"match and tobacco feched");

syscall(333,"smoker with paper is smoking\n");

//printf("paper smoking\n");

sleep(0.5);

sem\_post (&more\_needed); /\* signal to agent \*/

}

else sem\_post (&tobacco); /\* drop the match \*/

}

}

int main() {

pthread\_t th\_1, th\_2, th\_3, th\_4;

sem\_init(&more\_needed,0,1);

sem\_init(&match,0,0);

sem\_init(&paper,0,0);

sem\_init(&tobacco,0,0);

pthread\_create(&th\_1,NULL,agent,NULL); // Here 6 threads equals to 6 cars on the road.

pthread\_create(&th\_2,NULL,smoker\_with\_tobacco,NULL); // 3 cars are on the North road ready to move to south

pthread\_create(&th\_3,NULL,smoker\_with\_paper,NULL); // 3 cars are on the east road ready to move to west

pthread\_create(&th\_4,NULL,smoker\_with\_match,NULL);

pthread\_join(th\_1,NULL);

pthread\_join(th\_2,NULL);

pthread\_join(th\_3,NULL);

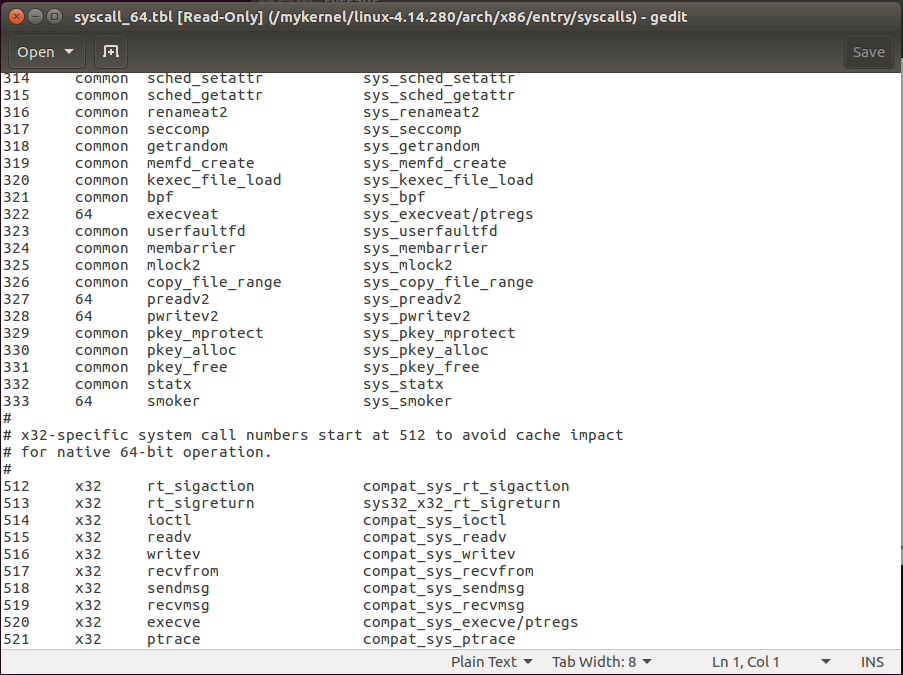
pthread\_join(th\_4,NULL);

return 0;

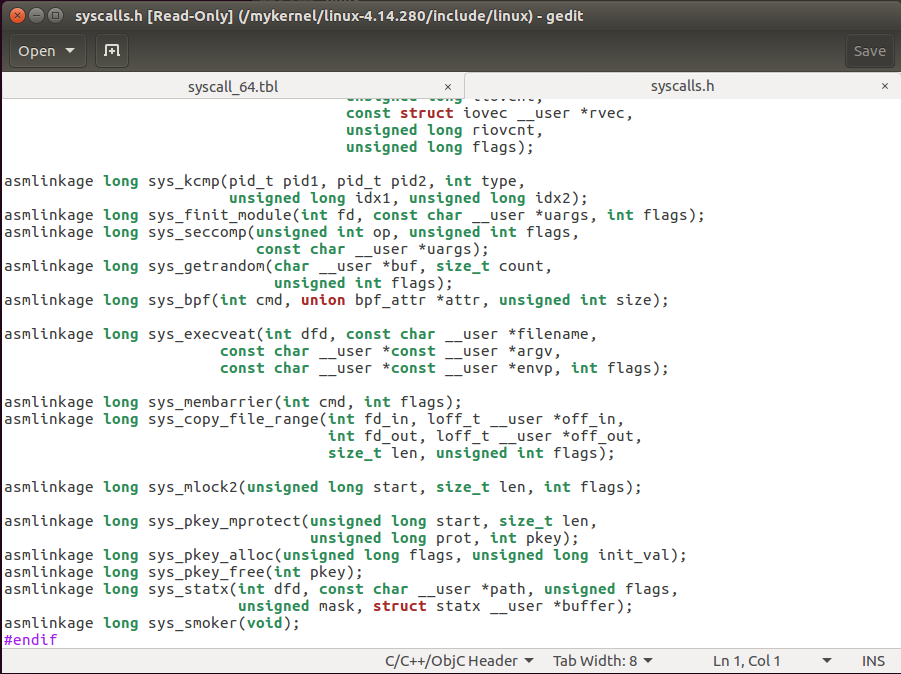
}

**KERNEL FILES MODIFICATION**

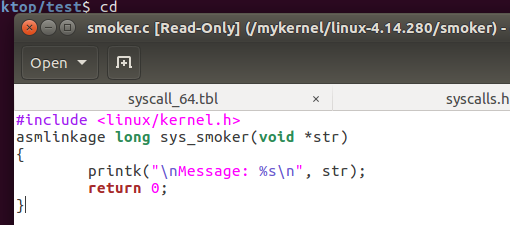
Path: gedit arch/x86/entry/syscalls/syscall\_64.tbl



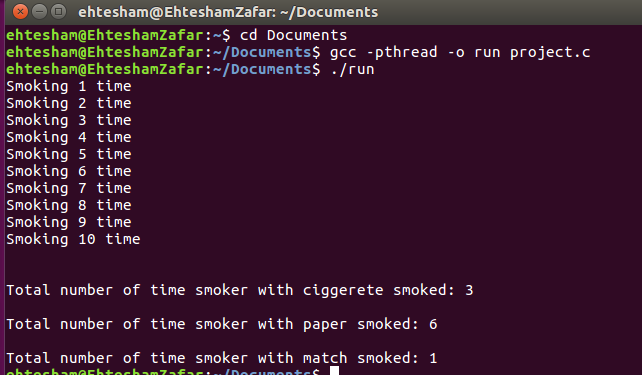
Path: gedit include/linux/syscalls.h

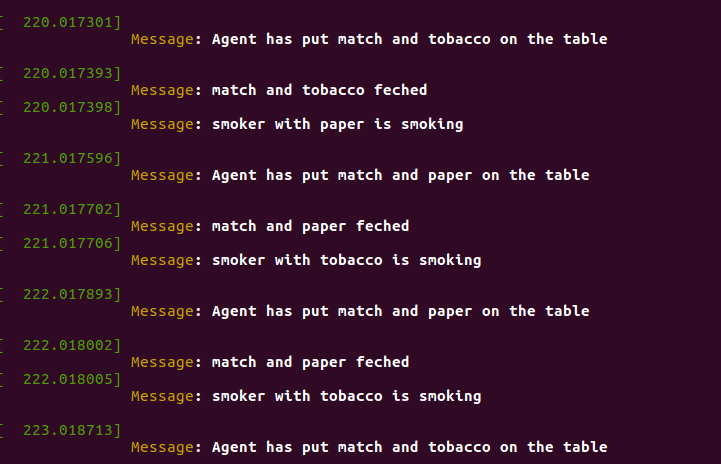


**KERNEL LEVEL CODE:**



**EXECUTION STATE:**





**LIMITATIONS AND DEADLOCK HANDLING**

The key claim of the cigarette smokers problem is that this scenario has no solution for traditional semaphores, as they existed at the time. When this problem was initially proposed, semaphores only provided operations for incrementing or decrementing their internal value by one. The problem proves that, if we are limited to those operations only, there are situations in which avoiding deadlock is provably impossible. Regardless of how the agent and the smoker threads are constructed, once the agent’s structure is fixed, any construction of the smokers will create a possible deadlock situation.

We could generalize the cigarette smokers problem to more than three threads. In this generalized form, there would be N smokers and the agent would place only N-1 items on the table. If every thread requires two resources (decrementing two semaphores, acquiring two locks, etc.), then a linear ordering will not prevent deadlock. The total number of available resources must be at least the total number of possible requests that can be made. If there are N threads that can all issue concurrent requests, there must be N instances available for the linear ordering to prevent deadlock.

**CONCULSION**

In the end our team efforts paid off and we were able to provide a solution to avoid deadlock in the first place that occurs in the chain smokers problem. This system call is essentially free of race condition, and is a demonstration of how the operating system avoids deadlock in the vast number of processes.

**GITHUB REPOSITORIES**

https://github.com/Syed007Hassan/Chain-Smoker-Problem-Using-Semaphores