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

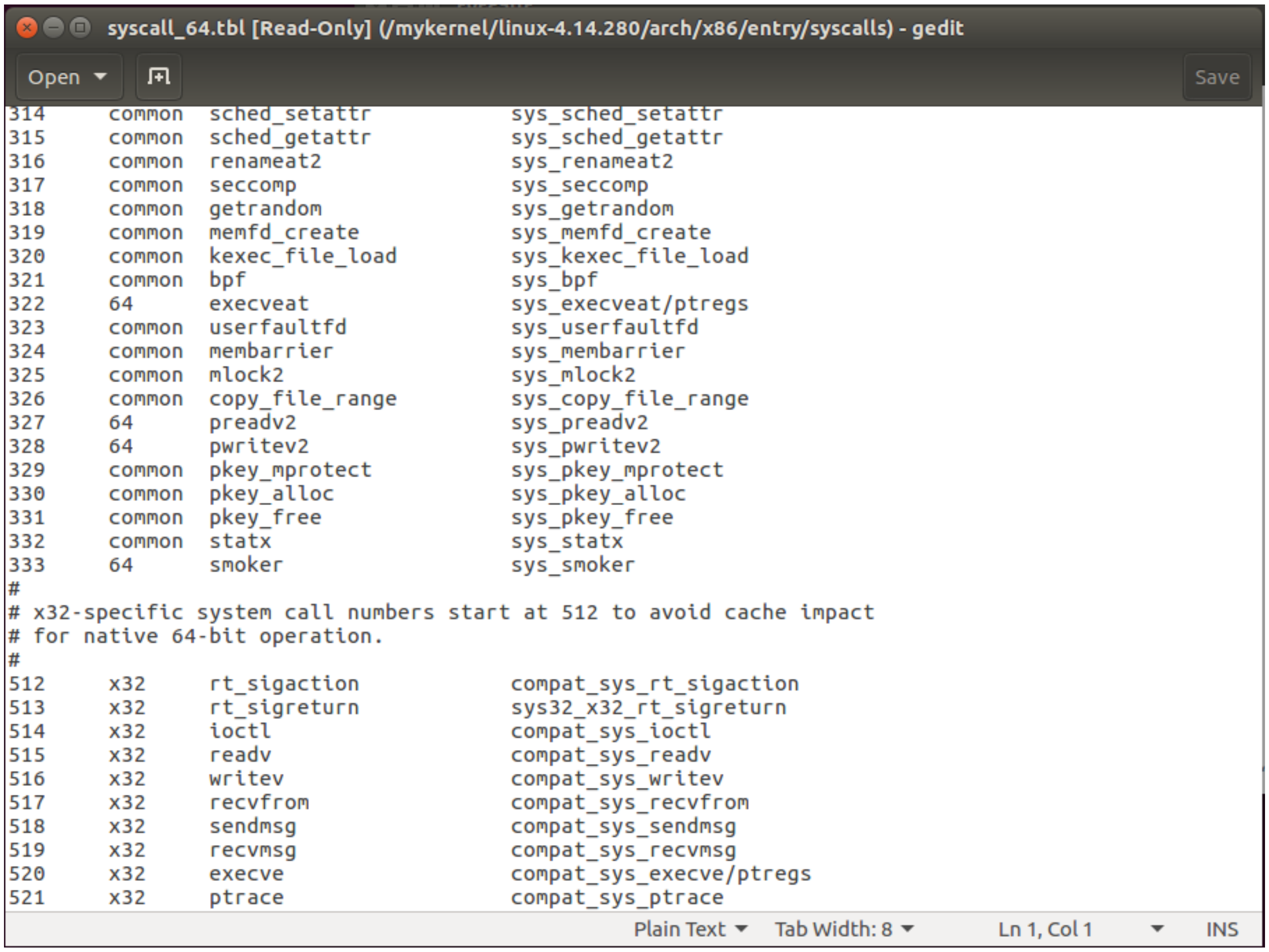
```
#include <stdio.h>
#include <unistd.h> /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <errno.h> /* Errors */
#include <stdio.h> /* Input/Output */
#include <stdlib.h> /* General Utilities */
#include <pthread.h> /* POSIX Threads */
#include <string.h> /* 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;

}
```

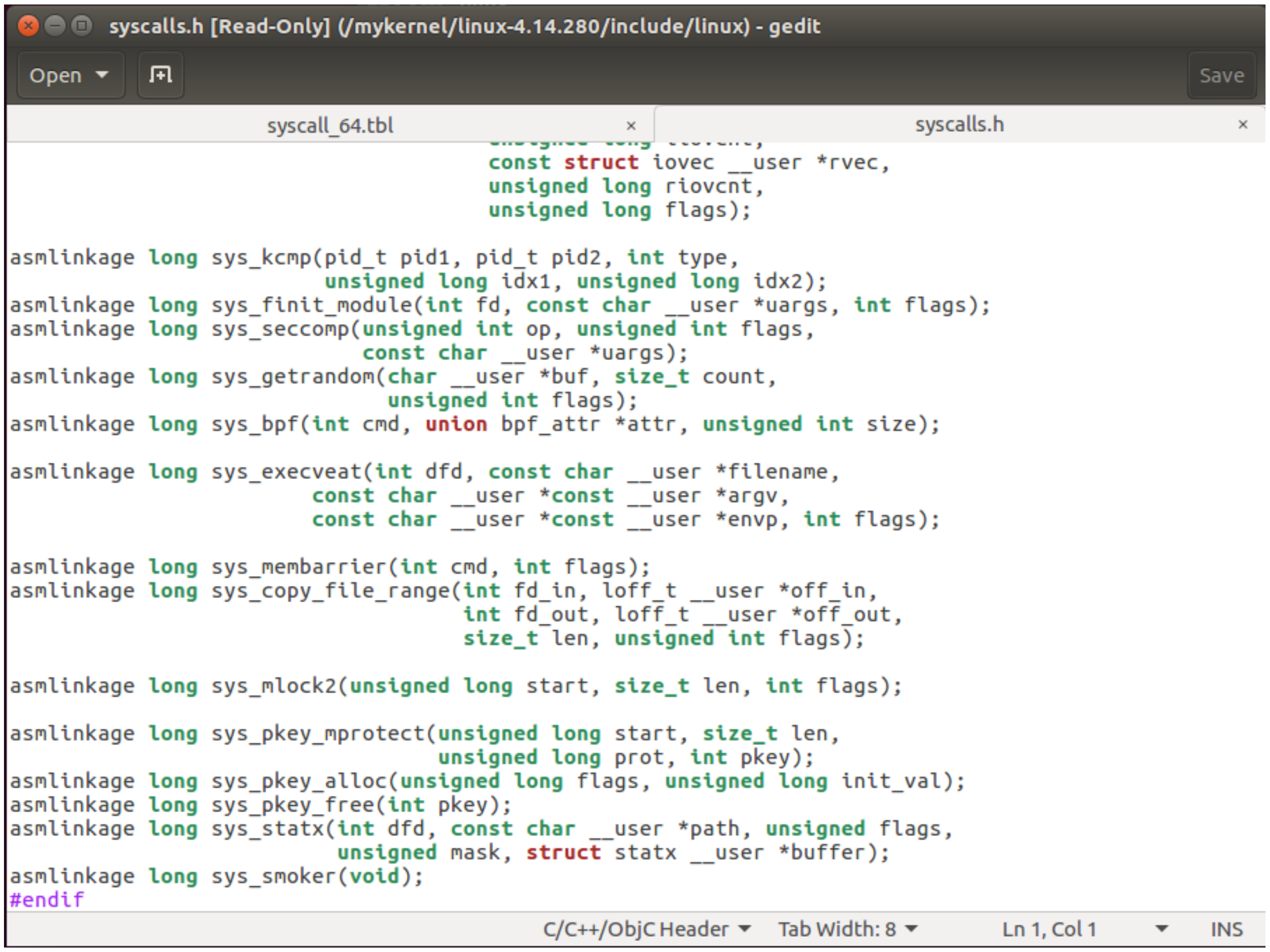
KERNEL FILES MODIFICATION

Path: gedit arch/x86/entry/syscalls/syscall_64.tbl


syscall_64.tbl [Read-Only] (/mykernel/linux-4.14.280/arch/x86/entry/syscalls) - gedit
Open
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

```
314     common  sched_setattr           sys_sched_setattr
315     common  sched_getattr           sys_sched_getattr
316     common  renameat2               sys_renameat2
317     common  seccomp                 sys_seccomp
318     common  getrandom               sys_getrandom
319     common  memfd_create            sys_memfd_create
320     common  kexec_file_load         sys_kexec_file_load
321     common  bpf                     sys_bpf
322     64      execveat                sys_execveat/ptregs
323     common  userfaultfd             sys_userfaultfd
324     common  membarrier              sys_membarrier
325     common  mlock2                  sys_mlock2
326     common  copy_file_range         sys_copy_file_range
327     64      preadv2                 sys_preadv2
328     64      pwritev2                sys_pwritev2
329     common  pkey_mprotect           sys_pkey_mprotect
330     common  pkey_alloc              sys_pkey_alloc
331     common  pkey_free               sys_pkey_free
332     common  statx                   sys_statx
333     64      smoker                  sys_smoker
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
513     x32     rt_sigreturn            sys32_x32_rt_sigreturn
514     x32     ioctl                   compat_sys_ioctl
515     x32     readv                   compat_sys_readv
516     x32     writev                  compat_sys_writev
517     x32     recvfrom                compat_sys_recvfrom
518     x32     sendmsg                 compat_sys_sendmsg
519     x32     recvmsg                 compat_sys_recvmsg
520     x32     execve                  compat_sys_execve/ptregs
521     x32     ptrace                  compat_sys_ptrace
```

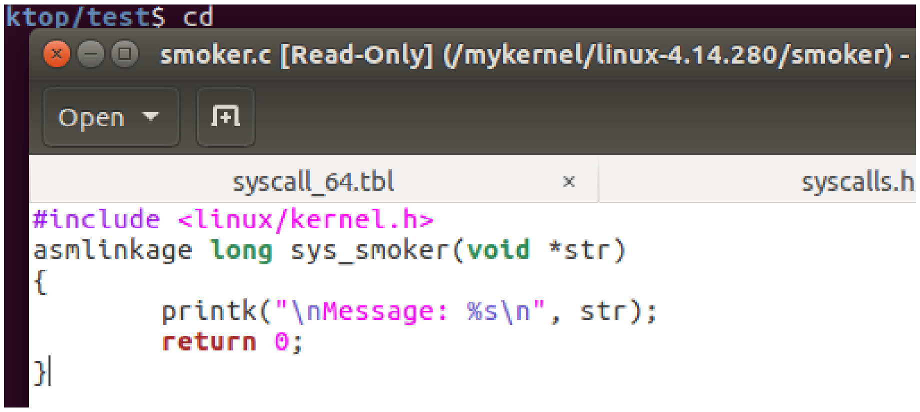
Path: gedit include/linux/syscalls.h



syscalls.h [Read-Only] (/mykernel/linux-4.14.280/include/linux) - gedit

```
                                   const struct iovec __user *rvec,
                                   unsigned long riovcnt,
                                   unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);
asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
                                    int fd_out, loff_t __user *off_out,
                                    size_t len, unsigned int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                  unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                          unsigned mask, struct statx __user *buffer);
asmlinkage long sys_smoker(void);
#endif
```

KERNEL LEVEL CODE:

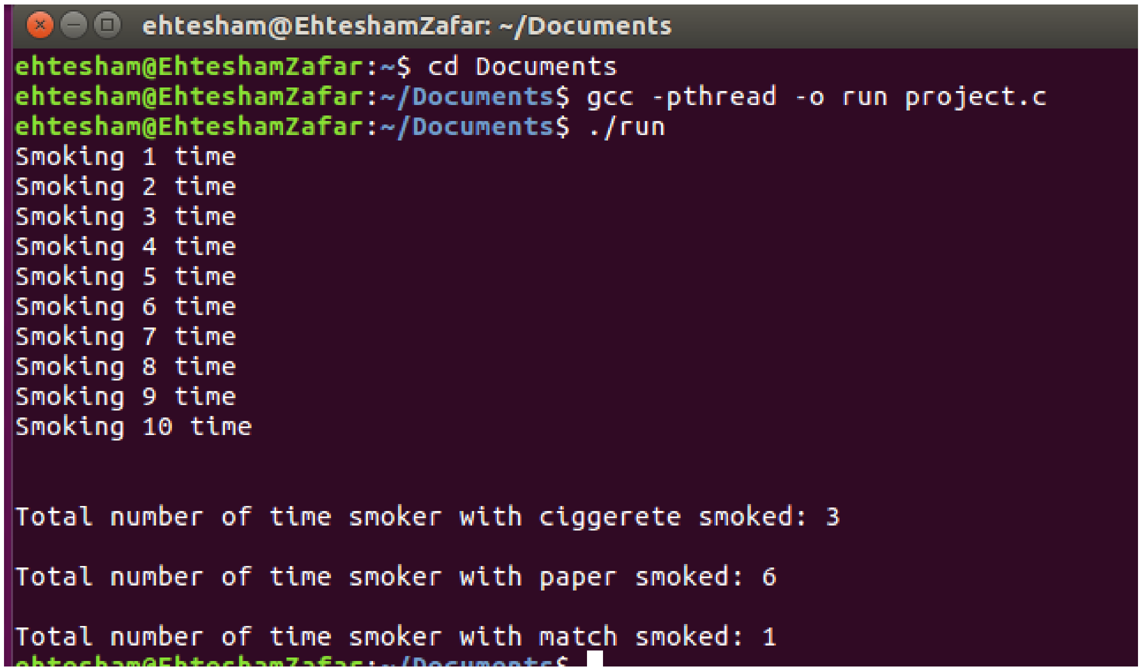


```
ktop/test$ cd
```

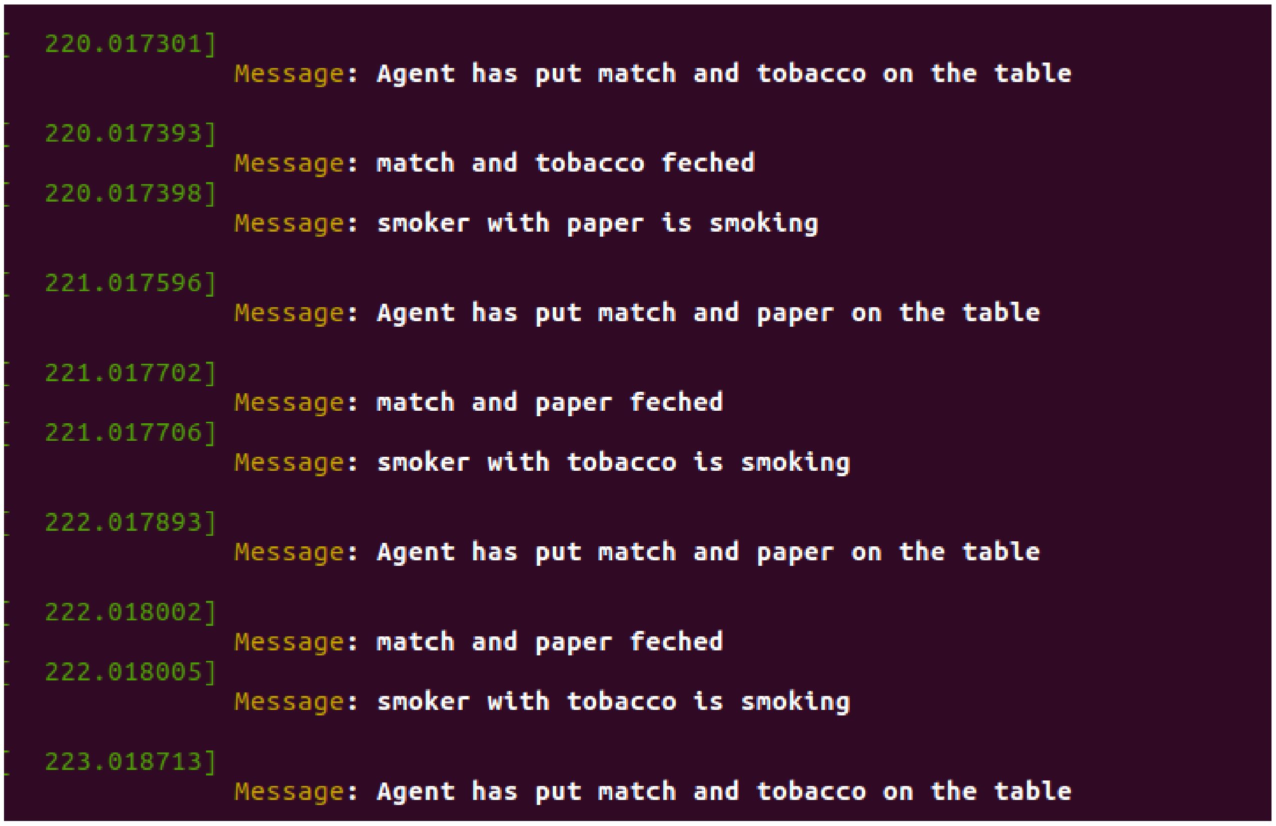
smoker.c [Read-Only] (/mykernel/linux-4.14.280/smoker) -

```
#include <linux/kernel.h>
asmlinkage long sys_smoker(void *str)
{
        printk("\nMessage: %s\n", str);
        return 0;
}
```

EXECUTION STATE:



```
ehtesham@EhteshamZafar: ~/Documents
ehtesham@EhteshamZafar:~$ cd Documents
ehtesham@EhteshamZafar:~/Documents$ gcc -pthread -o run project.c
ehtesham@EhteshamZafar:~/Documents$ ./run
Smoking 1 time
Smoking 2 time
Smoking 3 time
Smoking 4 time
Smoking 5 time
Smoking 6 time
Smoking 7 time
Smoking 8 time
Smoking 9 time
Smoking 10 time


Total number of time smoker with ciggerete smoked: 3

Total number of time smoker with paper smoked: 6

Total number of time smoker with match smoked: 1
```



```
[  220.017301]
               Message: Agent has put match and tobacco on the table

[  220.017393]
               Message: match and tobacco feched
[  220.017398]
               Message: smoker with paper is smoking

[  221.017596]
               Message: Agent has put match and paper on the table

[  221.017702]
               Message: match and paper feched
[  221.017706]
               Message: smoker with tobacco is smoking

[  222.017893]
               Message: Agent has put match and paper on the table

[  222.018002]
               Message: match and paper feched
[  222.018005]
               Message: smoker with tobacco is smoking

[  223.018713]
               Message: Agent has put match and tobacco on the table
```

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/EhteZafar/Chain-Smoker-Problem-