

**National University of Computer and Emerging Sciences**



Synchronized Chain Smokers: Addressing Concurrency Challenges

**Group Members**

Huzefa Tayyab…………22k-4188

Sarim Shah…………22k-4299

Muhammad Hasheem Iftikhar…………22k-4350

**Supervised by**

**Anaum Hamid**

**FAST School of Computing**

**National University of Computer and Emerging Sciences**

**Karachi, Pakistan**

**May 2024**

**1. Introduction**

The project is intended for the Operating Systems course, with the aim of implementing a solution to the synchronized chainsmoker problem. The purpose of this project is to implement a solution to the synchronized chainsmoker problem, a classic concurrency problem in operating systems. The scope includes designing and implementing a system involving an agent and three smokers, utilizing custom syscalls integrated into the Ubuntu Linux kernel for synchronization.

**Problem description:**

The synchronized chainsmoker problem, also known as the cigarette smokers problem, is a classic concurrency problem in operating systems. It involves multiple threads (smokers) and an agent, all competing for resources (tobacco, paper, and a match) to complete a task (smoking). The challenge lies in coordinating the actions of these threads to avoid deadlocks and ensure correct synchronization. In this scenario, there are three smokers and one agent. Each smoker has an infinite supply of one item (tobacco, paper, or a match) but needs the other two to roll and smoke a cigarette. The agent has access to all three items and is responsible for placing two items on the table for the smokers to use. A deadlock can occur if the synchronization between the agent and the smokers is not properly managed. For example, if the agent places two items on the table but is unable to signal to the appropriate smoker to pick them up, the system may deadlock, with the agent and smokers waiting indefinitely for each other to act.

**Solution implemented:**

In this project, custom syscalls are integrated into the Ubuntu Linux kernel to facilitate communication between the agent and the smokers. Kernel level signals are used for this which are basically Semaphores. Through semaphores, the agent signals the relevant smoker that items are available, the smoker then consumes the resources provided after which it signals the agent that it is done with smoking so that the agent can continue to place resources again on the table.

* 1. **Features.**

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.

Custom system calls are added to the kernel through recompilation which are then called through a user level program which is basically the application program that calls the system calls.

**Some notable features:**

**Syscall Definition:** Defines custom syscalls AGENT\_SYSCALL\_NUM and SMOKER\_SYSCALL\_NUM with specific syscall numbers (548 agent- 549 smoker)

**Kernel Space Implementation:** Implements the behavior of the custom syscalls in kernel space, involving synchronization mechanisms such as semaphores to solve the synchronized chainsmoker problem.

**Error Handling:** Handles errors gracefully, potentially returning error codes if the syscalls encounter issues during execution.

**Integration with Kernel:** The syscall program is integrated into the Ubuntu Linux kernel and recompiled to ensure proper functioning of the custom syscalls.

**Concurrency Control:** Implements synchronization mechanisms such as semaphores to prevent deadlock and ensure correct behavior of the synchronized chainsmoker system.

**Interprocess Communication:** Facilitates communication between the user space program and the kernel space implementation through the invocation of custom syscalls.

* 1. **technology used.**

Programming Language: C language

Oracle VM Virtualbox 7.0.14

Operating system: Ubuntu 16.04 LTS

Kernel recompiled: linux 4.17.4

* 1. **code snippets.**

**Kernel level syscalls:**

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/syscalls.h>

#include <linux/sched.h>

#include <linux/delay.h>

#include <linux/semaphore.h>

#include <linux/completion.h> // Include completion header

// Define semaphores for synchronization

DEFINE\_SEMAPHORE(agent\_sem);

DECLARE\_COMPLETION(smoker\_comp[3]); // Declare completions for synchronization

asmlinkage long agent\_syscall(int \_randomNum) {

int select;

select =\_randomNum;

down\_interruptible(&agent\_sem);

if (select == 0) {

printk(KERN\_INFO "Agent placed tobacco and paper on the table\n");

} else if (select == 1) {

printk(KERN\_INFO "Agent placed tobacco and matches on the table\n");

} else if(select==2){

printk(KERN\_INFO "Agent placed paper and matches on the table.\n");

}

else{printk(KERN\_INFO "Agent does not place anything.\n");}

if(select>-1 && select<3)

{

complete(&smoker\_comp[select]); // Signal the corresponding smoker

}

return 0;

}

asmlinkage long smoker\_syscall(int smokeid) {

wait\_for\_completion\_interruptible(&smoker\_comp[smokeid]); // Wait for the agent to signal

printk(KERN\_INFO "Smokers are ready\n");

if (smokeid == 0) {

printk(KERN\_INFO "Smoker with matches sat on table\n");

} else if (smokeid == 1) {

printk(KERN\_INFO "Smoker with paper sat on table\n");

} else {

printk(KERN\_INFO "Smoker with tobacco sat on table\n");

}

msleep(2000); // Sleep for 2 seconds

switch (smokeid) {

case 0:

printk(KERN\_INFO "Smoker %d with matches smoking\n", smokeid);

break;

case 1:

printk(KERN\_INFO "Smoker %d with paper smoking\n", smokeid);

break;

case 2:

printk(KERN\_INFO "Smoker %d with tobacco smoking\n", smokeid);

break;

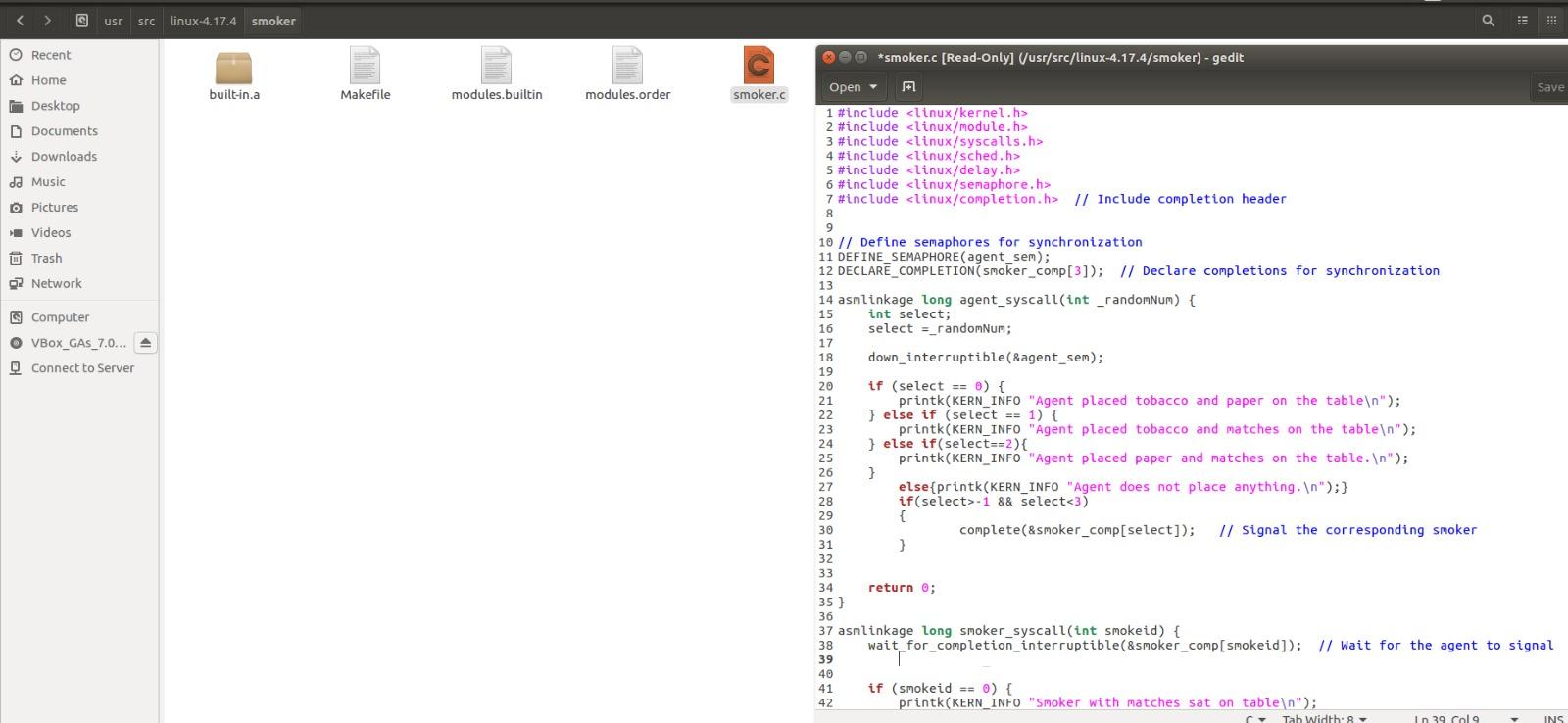
}

msleep(3000); // Sleep for 3 seconds

up(&agent\_sem); // Release the agent semaphore

return 0;

}

****

**User space program:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <sys/syscall.h>

#include <signal.h>

#define AGENT\_SYSCALL\_NUM 548

#define SMOKER\_SYSCALL\_NUM 549

int main() {

int randomNum;

int smokeid;

char choice;

// Call three smokers together

for (int i = 0; i < 3; ++i) {

long ret\_smoker = syscall(SMOKER\_SYSCALL\_NUM, i);

if (ret\_smoker != 0) {

printf("Error in executing smoker\_syscall.\n");

return 1;

}

}

// Activate the agent

long ret\_agent = syscall(AGENT\_SYSCALL\_NUM, 0);

if (ret\_agent != 0) {

printf("Error in executing agent\_syscall.\n");

return 1;

}

srand(time(NULL));

// Send random numbers to the agent until Ctrl+C is pressed

while (1) {

randomNum = rand();

long ret\_agent = syscall(AGENT\_SYSCALL\_NUM, randomNum);

if (ret\_agent != 0) {

printf("Error in executing agent\_syscall.\n");

return 1;

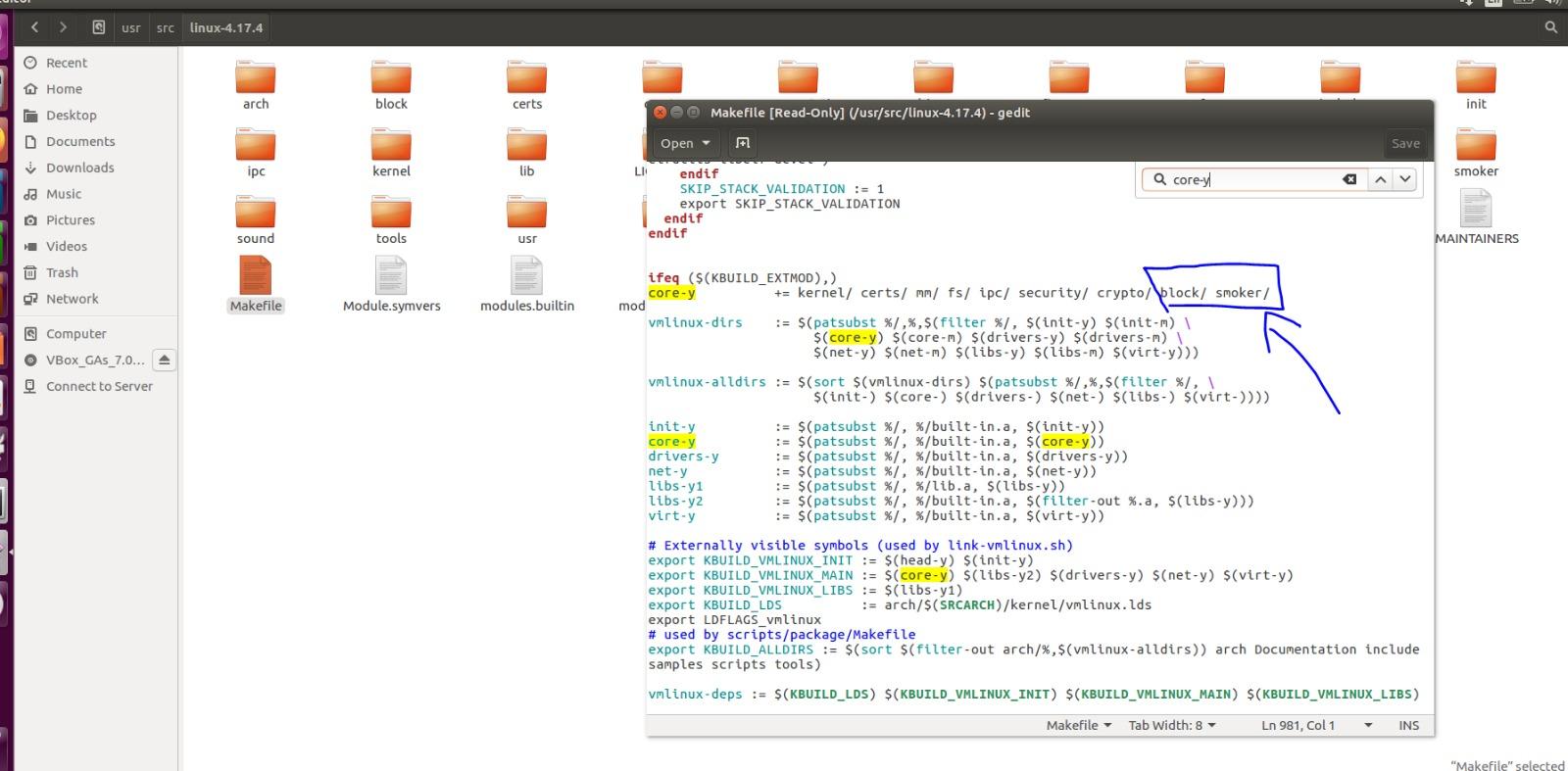
}

}

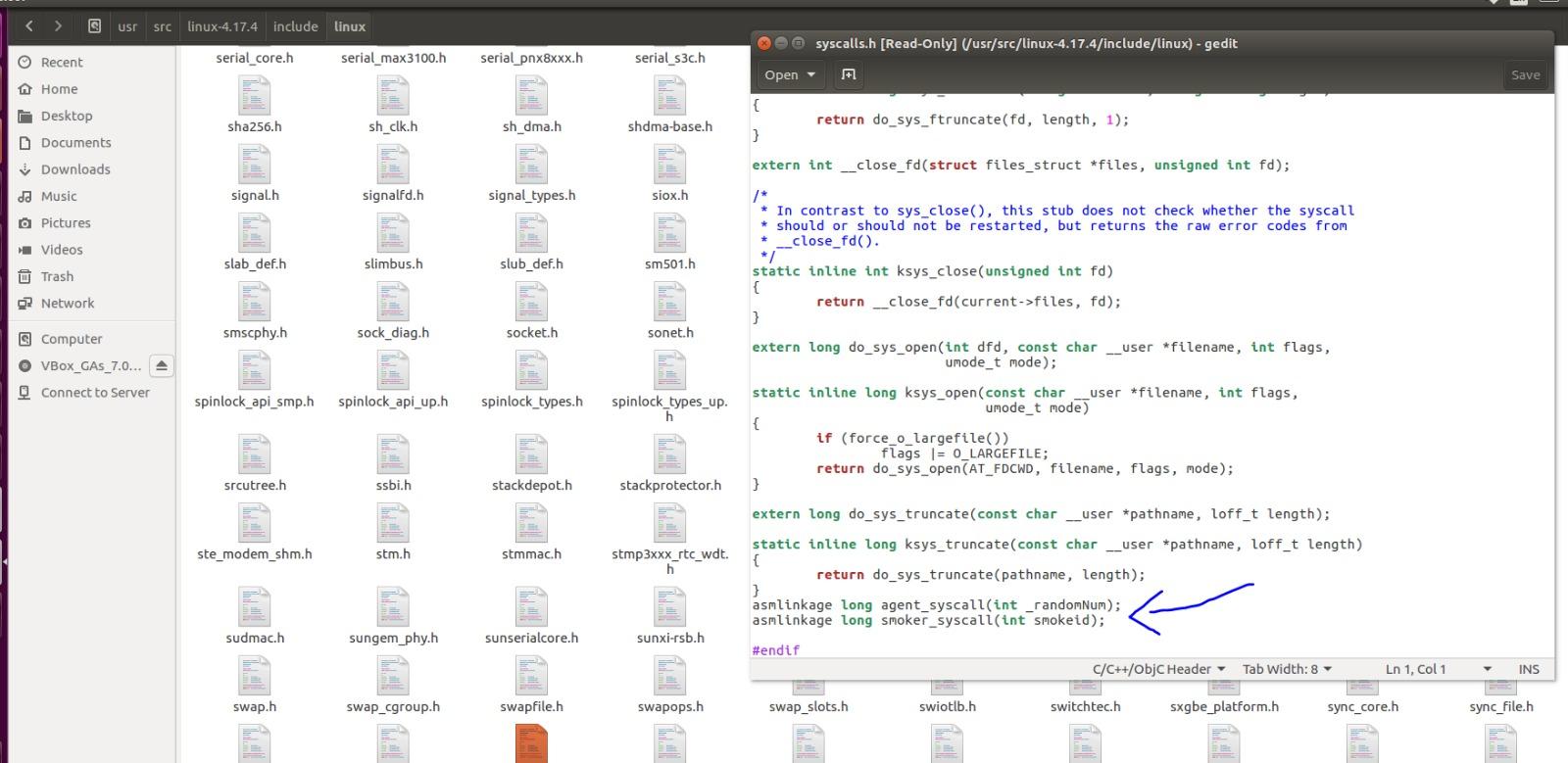
return 0;

}

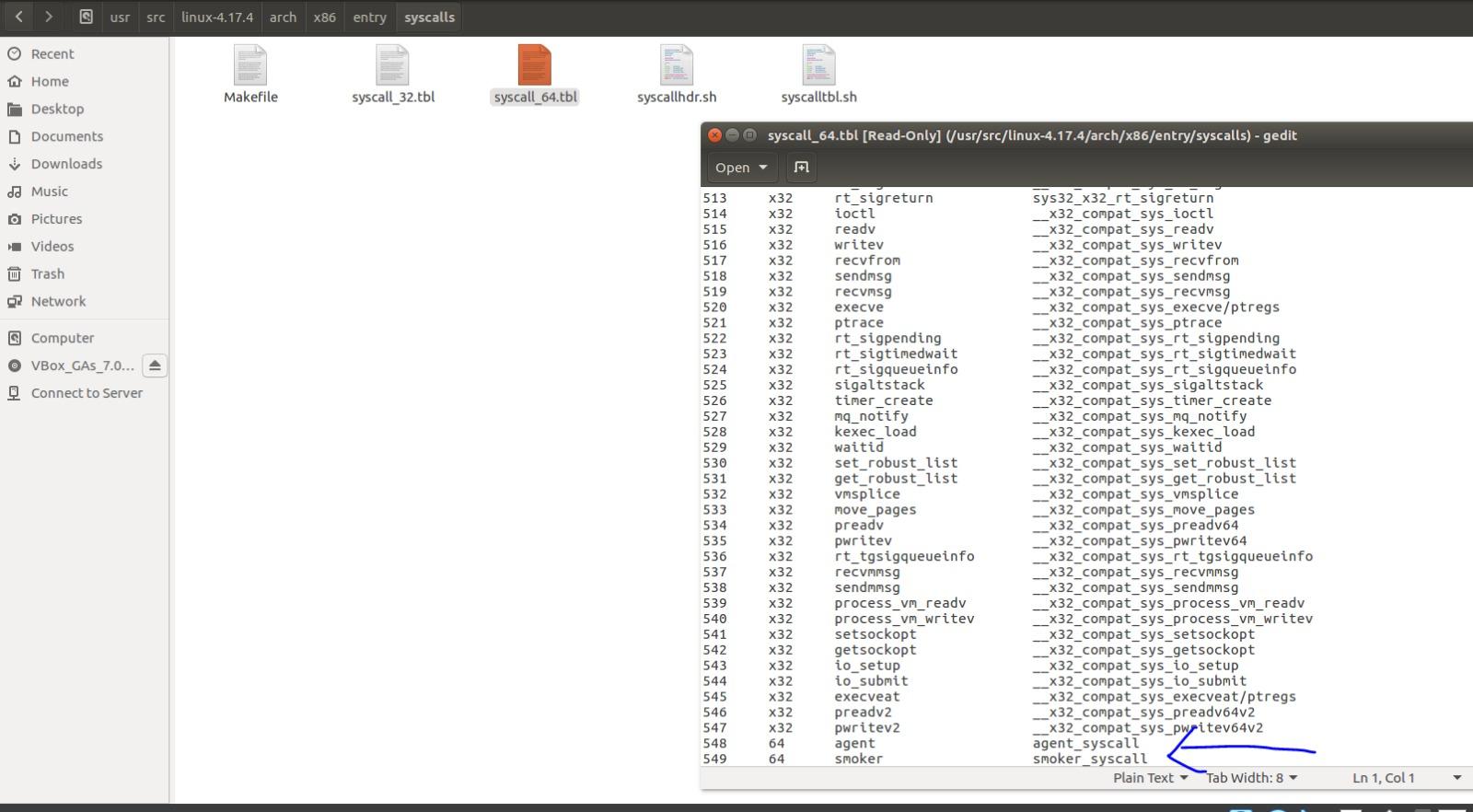
**linux makefile core-y:**

****

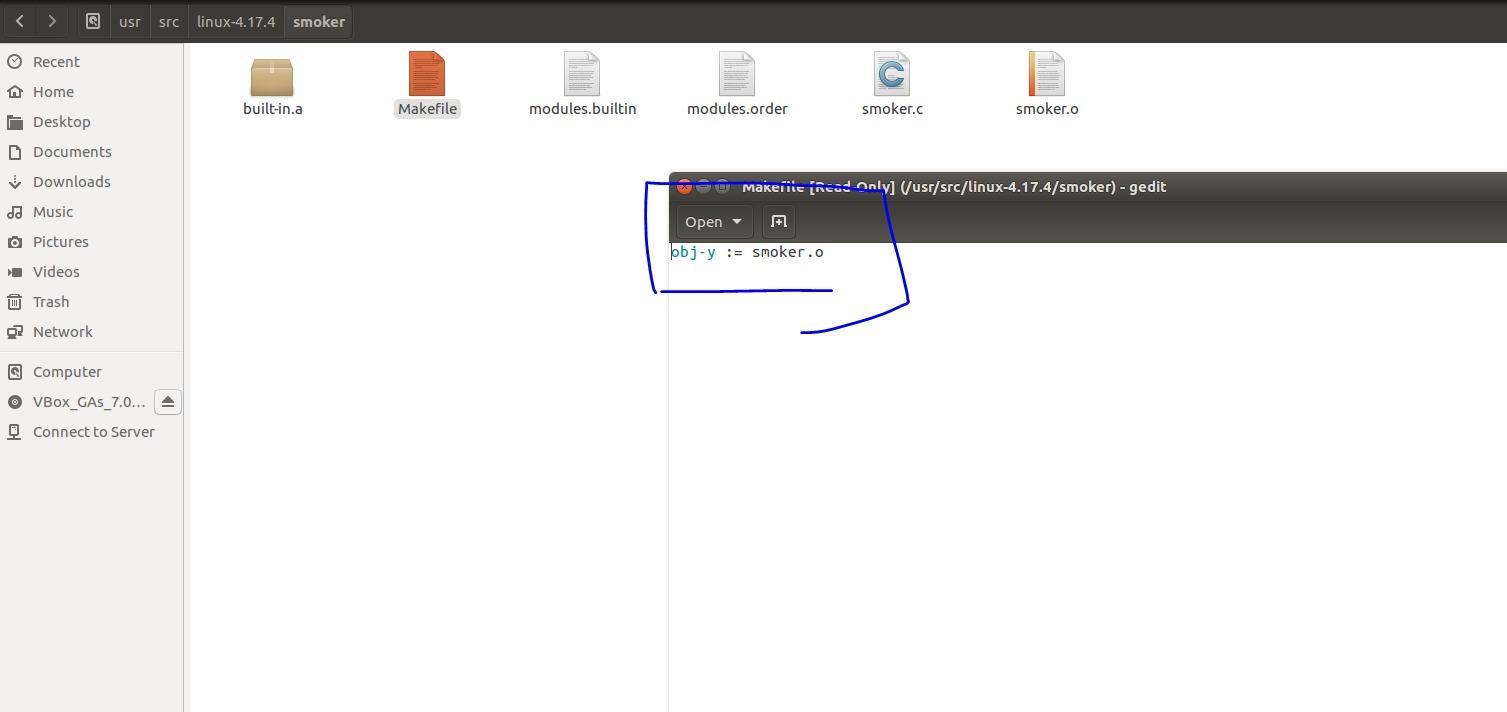
**Syscalls.h kernel level file:**

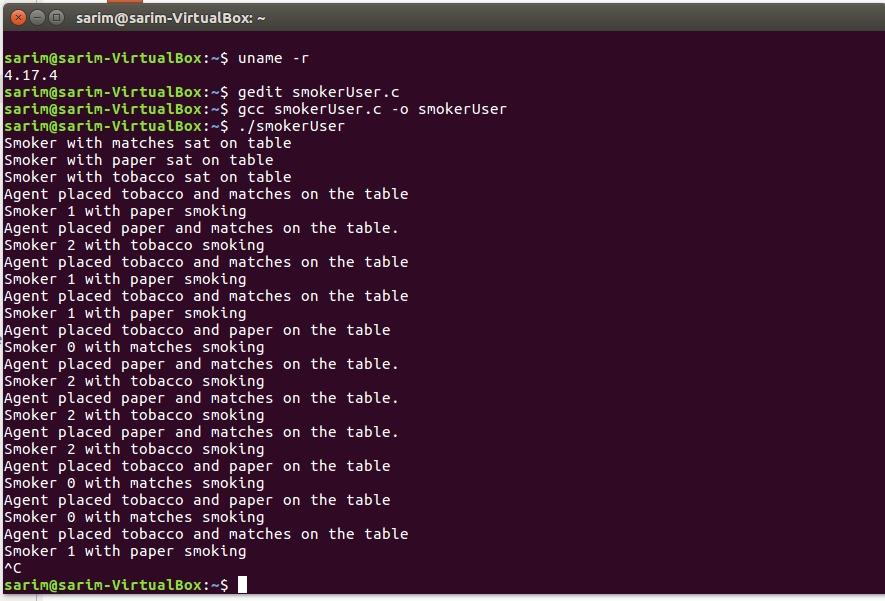
****

**Syscall table update:**

****

*agent 548 smoker 549*

**Smoker makefile:**

* 1. **Output **

**6. Limitations of this problem:**

The core assertion of the cigarette smokers problem is its inherent deadlock challenge under traditional semaphore implementations. Initially proposed when semaphores could only increment or decrement their internal value by one, the problem illustrates the impossibility of deadlock avoidance within certain constraints. Regardless of the threading architecture, if the agent's behavior remains fixed, any smoker configuration may lead to deadlock.

Expanding the problem to involve more than three threads introduces further complexity. With N smokers and the agent placing only N-1 items on the table, deadlock remains a risk. Each thread requiring two resources implies that a linear ordering cannot prevent deadlock. To avoid deadlock, the total number of available resources must equal or exceed the total number of potential concurrent requests. If N threads can all request resources concurrently, there must be N resource instances to maintain a linear ordering and prevent deadlock.

**7. Conclusion.**

In conclusion, the synchronized chainsmoker project has progressed smoothly and successfully completed. The implementation of custom syscalls integrated into the Ubuntu Linux kernel has provided an effective solution to the synchronized chainsmoker problem, demonstrating successful concurrency control and interprocess communication.

Throughout the project, we have successfully addressed the challenges of coordinating the actions of the agent and smokers to avoid deadlocks and ensure correct synchronization. By utilizing semaphores and careful design considerations, we have created a robust system that allows the smokers to acquire the necessary resources and smoke cigarettes without encountering conflicts or race conditions.

**References** :

<https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872>