# CS–205 OPERATING SYSTEMS

## "PROJECT REPORT"

***PROJECT TITLE:***

**"System Call for Semaphore (Example: Reader- Writer Problem)"**

***GROUP MEMBERS:***


1. MURAD POPATTIA (17K-3722)
2. MUHAMMAD SAMIULLAH (17K-3745)
3. SAFIULLAH KHARADI (17K-3737)


***GROUP SECTION:*** B

***LANGUAGE:*** C++ / C

***OPERATING SYSTEM:*** Ubuntu 18.04

***COURSE INSTRUCTOR:*** MISS NAUSHEEN SHOAIB

# READERS-WRITERS PROBLEM:

In the readers-writers problem, there is a critical section that both reader and writer can access. Reader only reads from the data while writer can both read and write to the data. More than one reader can read at the same time. A writer cannot access the data while a reader is reading. No other thread can access the memory while a writer is accessing the data.

# PROPOSED SOLUTION:

To solve this situation, a writer should get exclusive access to an object that is:

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However if some person is reading the file, then others may read it at the same time.

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource. Here priority means, no reader should wait if the share is currently opened for reading.

Some problem parameters to acknowledge include:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read

# ALGORITHM USED:

## WRITER FUCNTION:

```
do {
    // writer requests for critical section
    wait(wrt);

    // performs the write

    // leaves the critical section
    signal(wrt);

} while(true);
```

## *READER FUNCTION:*

```
do {

   // Reader wants to enter the critical section
   wait(mutex);

   // The number of readers has now increased by 1
   readcnt++;

   // there is atleast one reader in the critical section
   // this ensure no writer can enter if there is even one reader
   // thus we give preference to readers here
   if (readcnt==1)
      wait(wrt);

   // other readers can enter while this current reader is inside
   // the critical section
   signal(mutex);

   // current reader performs reading here
   wait(mutex);   // a reader wants to leave

   readcnt--;

   // that is, no reader is left in the critical section,
   if (readcnt == 0)
       signal(wrt);         // writers can enter

   signal(mutex); // reader leaves

} while(true);
```

# *GOAL:*

The goal of this assignment is to implement a multithreaded solution for the above mentioned algorithm using Semaphores. Moreover, steps are also taken to calculate the average time of between a request and an entry into the critical section by the reader or writer.

# *APPROACH USED:*

1. Make a system call that simulates the readers/writers problem with a hard coded number of readers and writers.
2. Use of average time as a parameterized system call which calculates and prints the average time between a request and an entry into the critical section listed above on kernel space.

## *Times:*

- ***Request Time:***
  This is the time when the reader / writer as requested for getting access to the critical section i.e. to perform their operation.
- ***Entry Time:***
  This is the time when the reader / writer achieves the permission and starts their work of reading / writing.
- ***Exit Time:***
  This is the time when the reader / writer finished their work and decides to leave.

# *APPROACH # 01:*

## *Library used:*

- **semaphore.h :** to add the semaphore
- **iostream :** for basic std functions
- **atomic :** added for time calculations
- **stdio.h :** added for integrating c functions
- **ctime :** added for time calculations and sleep()
- **chrono :** added for time calculations
- **pthread.h :** added for multithreading
- **fstream :** added for filing support
- **unistd.h :** added for size_t and various other functions

## *Semaphores:*

- mutex - binary semaphore for writer entry into critical section.
- rwmutex - binary semaphore for shared variable read_count safety.
- avgmutex - binary semaphore for shared variable avg_time safety.

## *Global Variables:*

- read_count - number of readers inside critical section: initialised to 0
- avgtime - average time taken for each thread to access the CS
- wrand_int - random time for writer to be in critical section

## *Functions for semaphore:*

- **sem_wait()** - decrements the semaphore value.
- **sem_post ()** - increments the semaphore value.

## *Reader Function:*

1. Reader requests the entry to critical section.

2. If allowed:
   - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **mutex** semaphore to restrict the entry of writers if any reader is inside.
   - It then, signals **rwmutex** as any other reader is allowed to enter while others are already reading.
   - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore **mutex** as now, writer can enter the critical section.

3. If not allowed, it keeps on waiting.
4. timespec struct used in order to calculate times between request and entries.

Thus, the semaphore **mutex** is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

## *Writer Function:*

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.
4. timespec struct used in order to calculate times between request and entries.

## *Functions for times:*

- chrono::system_clock::now() - inorder to get recent system time
- sleep() - inorder to simulate the waiting by adding a delay

## *Program code(.cpp):*

```
#include <semaphore.h>
#include <iostream>
#include <atomic>
#include <stdio.h>
#include <ctime>
#include <chrono>
#include <pthread.h>
#include <fstream>
#include <unistd.h>

using namespace std;

struct read_info{
      int id;
      int time;
};

void *writer(void *i);      // writer thread
void *reader(void *i);       // reader threads
int wrand_int;

int avg_time;

sem_t mutex;            // semaphore for writer entry into cs
sem_t rwmutex;          // semaphore for shared variable read_count safety
sem_t avgmutex;         // semaphore for shared variable avg_time safety
int read_count=0;       // number of readers inside cs


int main()
{
    // initialising semaphores
    sem_init(&mutex,0,1);
    sem_init(&rwmutex,0,1);
    sem_init(&avgmutex,0,1);

    int nw,nr;
    ifstream inFile ;
    inFile.open("input.txt");    //we read from the input file
    inFile>>nr;
    inFile>>nw;

    srand(time(0));
    // creating n pthreads
    pthread_t w[nw],r[nr];
    pthread_attr_t w_attr[nw],r_attr[nr];
    struct read_info read[nr];

      for (int i = 0; i < nw; ++i) {
            pthread_attr_init(&w_attr[i]);

      pthread_create(&w[i],&w_attr[i],writer,(void*)(intptr_t)(i+1));
```

```
      }

       for (int i = 0; i < nr; ++i) {
            read[i].time=(rand()%4)+1;
            read[i].id=i+1;
            pthread_attr_init(&r_attr[i]);
            pthread_create(&r[i],&r_attr[i],reader,&read[i]);
      }


    // joining the threads
       for(int i=0;i<nw;i++)
                pthread_join(w[i],NULL);
    for(int i=0;i<nr;i++)
                pthread_join(r[i],NULL);


    // destroying semaphores
    sem_destroy(&mutex);
    sem_destroy(&rwmutex);
    sem_destroy(&avgmutex);
      syscall(336,avg_time,nr,nw); // system call made here
      printf("\nAvg time spent by readers and writers in critical
section(sec): %d\n",(avg_time/((nw+nr)*1000))/1000);

    return 0;

}

void *writer(void * param){

      int id = (intptr_t)param;

        // calculating request time
        auto reqTime = std::chrono::system_clock::now();
        time_t my_time;
        time (&my_time);
        struct tm *timeinfo = localtime (&my_time);
      printf("Request by Writer Thread %d at %02d:%02d\n",id,timeinfo-
>tm_min,timeinfo->tm_sec);


        sem_wait(&rwmutex); // wait writer

        // calculating entery time
        auto enterTime = std::chrono::system_clock::now();
        time (&my_time);
        timeinfo = localtime (&my_time);
      wrand_int=(rand()%4)+1;
      printf("\tEntry by Writer Thread %d at %02d:%02d | Will take t = %d
sec to write\n",id,timeinfo->tm_min,timeinfo->tm_sec,wrand_int);

        sleep(wrand_int);// simulate a thread executing in CS
```

```
        auto exitTime = std::chrono::system_clock::now();
        time (&my_time);
        timeinfo = localtime (&my_time);
      printf("\t\tExit by Writer Thread %d at %02d:%02d\n",id,timeinfo-
>tm_min,timeinfo->tm_sec);

        sem_post(&rwmutex);   // signal writer

        // adding waiting time to shared variable avg_time
        sem_wait(&avgmutex);
        avg_time +=
std::chrono::duration_cast<std::chrono::microseconds>(enterTime-
reqTime).count();
        sem_post(&avgmutex);
}

void *reader(void * param)
{

      struct read_info *ptr = (struct read_info*)param;

        auto reqTime = std::chrono::system_clock::now();
        time_t my_time;
        time (&my_time);
        struct tm *timeinfo = localtime (&my_time);
      printf("Request by Reader Thread %d at %02d:%02d\n",ptr-
>id,timeinfo->tm_min,timeinfo->tm_sec);


        sem_wait(&mutex);  // wait for read_count access permission
        read_count++;       // increment read count as new reader is
entering
        if(read_count==1)   // only if this is the first reader
            sem_wait(&rwmutex); // then wait for cs permission
        sem_post(&mutex);   // signal mutex

        auto enterTime = std::chrono::system_clock::now();
        time (&my_time);
        timeinfo = localtime (&my_time);
      printf("\tEntry by Reader Thread %d at %02d:%02d | Will take t = %d
sec to read\n",ptr->id,timeinfo->tm_min,timeinfo->tm_sec,ptr->time);

        sleep(ptr->time);// simulate a thread executing in CS

        sem_wait(&mutex);   // wait for read count access permission
        read_count--;       // decrement readcount as we are done reading
        if(read_count==0)   // only if this is the last reader
            sem_post(&rwmutex); // signal rwmutex


      auto exitTime = std::chrono::system_clock::now();
        time (&my_time);
        timeinfo = localtime (&my_time);
```

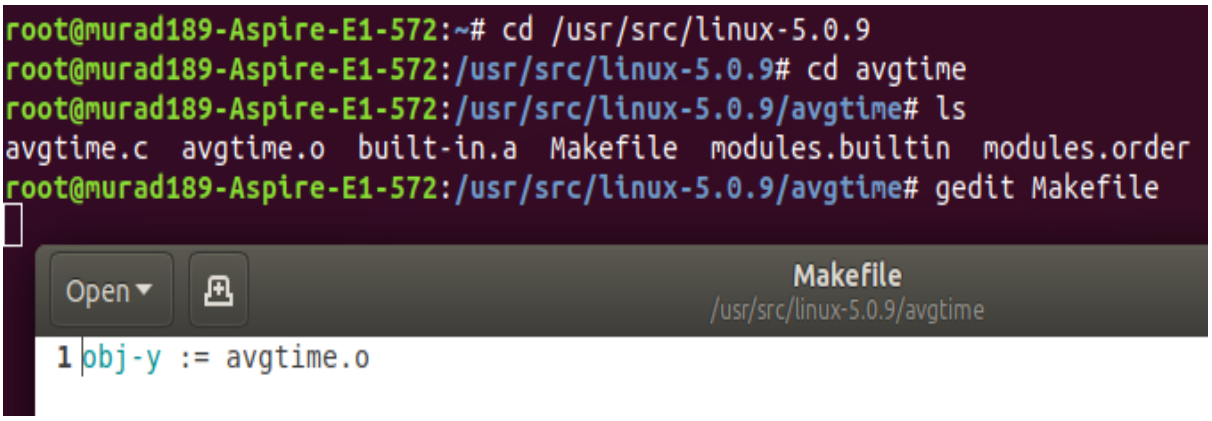
```
      printf("\t\tExit by Reader Thread %d at %02d:%02d\n",ptr-
>id,timeinfo->tm_min,timeinfo->tm_sec);

        sem_post(&mutex);
        sem_wait(&avgmutex);
        avg_time +=
std::chrono::duration_cast<std::chrono::microseconds>(enterTime-
reqTime).count();
        sem_post(&avgmutex);
}
```

## *Parameterized System call code (#337):*

```
#include<linux/kernel.h>
#include<linux/syscalls.h>

SYSCALL_DEFINE3(avgtime,int, avg_time,int, nw,int, nr){

      printk("Avg time taken for RW is %d
seconds\n",(avg_time/((nw+nr)*1000))/1000);
      return 0;
}
```

## *Screenshots:*

- This shows the Makefile for the system call stored in the c file ‘avgtime’



- This shows that the system call for ‘avgtime’ is added to Makefile of the kernel

```
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9# gedit Makefile
```

Makefile
/usr/src/linux-5.0.9

```
    SKIP_STACK_VALIDATION := 1
    export SKIP_STACK_VALIDATION
  endif
endif

PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ avgtime/ rwprob/

```

- This shows the linkage file for the system call

```
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9# cd include/linux
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/include/linux# gedit syscalls.h
```

syscalls.h
/usr/src/linux-5.0.9/include/linux

```
        return old;
}

asmlinkage long sys_hello(void);

asmlinkage long sys_avgtime(int,int,int);

asmlinkage long sys_rwprob(void);

#endif
```

- This shows that the system call has been added to the syscall_64 table

```
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/arch/x86/entry/syscalls# ls
Makefile  syscall_32.tbl  syscall_64.tbl  syscallhdr.sh  syscalltbl.sh
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/arch/x86/entry/syscalls# gedit
syscall_64.tbl
```

syscall_64.tbl
/usr/src/linux-5.0.9/arch/x86/entry/syscalls

```
330     common  pkey_alloc              __x64_sys_pkey_alloc
331     common  pkey_free               __x64_sys_pkey_free
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
335     64      hello                   sys_hello
336     common  avgtime                 __x64_sys_avgtime
337     64      rwprob                  sys_rwprob

```

# *Output:*

```
murad189@murad189-Aspire-E1-572: ~
File  Edit  View  Search  Terminal  Help
murad189@murad189-Aspire-E1-572:~$ g++ -o a backup.cpp -lpthread
murad189@murad189-Aspire-E1-572:~$ clear
murad189@murad189-Aspire-E1-572:~$ ./a
Request by Writer Thread 1 at 46:57
Request by Writer Thread 2 at 46:57
Request by Writer Thread 3 at 46:57
Request by Writer Thread 4 at 46:57
Request by Writer Thread 5 at 46:57
        Entry by Writer Thread 1 at 46:57 | Will take t = 4 sec to write
Request by Writer Thread 6 at 46:57
Request by Reader Thread 1 at 46:57
Request by Reader Thread 2 at 46:57
Request by Reader Thread 3 at 46:57
Request by Reader Thread 4 at 46:57
Request by Reader Thread 5 at 46:57
Request by Reader Thread 6 at 46:57
                Exit by Writer Thread 1 at 47:01
        Entry by Writer Thread 2 at 47:01 | Will take t = 1 sec to write
                Exit by Writer Thread 2 at 47:02
        Entry by Writer Thread 3 at 47:02 | Will take t = 2 sec to write
                Exit by Writer Thread 3 at 47:04
        Entry by Writer Thread 4 at 47:04 | Will take t = 4 sec to write
                Exit by Writer Thread 4 at 47:08
        Entry by Writer Thread 5 at 47:08 | Will take t = 4 sec to write
```

```
murad189@murad189-Aspire-E1-572: ~
File  Edit  View  Search  Terminal  Help
                Exit by Writer Thread 4 at 47:08
        Entry by Writer Thread 5 at 47:08 | Will take t = 4 sec to write
                Exit by Writer Thread 5 at 47:12
        Entry by Writer Thread 6 at 47:12 | Will take t = 3 sec to write
                Exit by Writer Thread 6 at 47:15
        Entry by Reader Thread 1 at 47:15 | Will take t = 4 sec to read
        Entry by Reader Thread 2 at 47:15 | Will take t = 1 sec to read
        Entry by Reader Thread 4 at 47:15 | Will take t = 1 sec to read
        Entry by Reader Thread 5 at 47:15 | Will take t = 3 sec to read
        Entry by Reader Thread 3 at 47:15 | Will take t = 1 sec to read
        Entry by Reader Thread 6 at 47:15 | Will take t = 4 sec to read
                Exit by Reader Thread 2 at 47:16
                Exit by Reader Thread 3 at 47:16
                Exit by Reader Thread 4 at 47:16
                Exit by Reader Thread 5 at 47:18
                Exit by Reader Thread 1 at 47:19
                Exit by Reader Thread 6 at 47:19

Avg time spent by readers and writers in critical section(sec): 12
murad189@murad189-Aspire-E1-572:~$ dmesg | tail -2
               Avg time spent by readers and writers in critical section(sec): 1
0
[ 2462.020784] Avg time taken for RW is 12 seconds
murad189@murad189-Aspire-E1-572:~$
```

# *APPROACH # 02:*

## *Library used:*

- **linux/init.h :** to accommodate the init module for testing
- **linux/semaphore.h :** to accommodate semaphores
- **linux/module.h :** because the initial system call was first tested as a kernel module
- **linux/kernel.h :** integral library to be used for making a system call
- **linux/kthread.h :** to make use of kthreads
- **linux/time.h :** to accomdate the time fuctions used such as msleep_interruptible()
- **linux/timer.h :** to accomdate for "time struct" used within the code
- **linux/delay.h** : to check if the kernel functions could be delayed
- **asm/delay.h :** to check if the kernel functions can be delayed
- **linux/random.h :** to allow for random generating of times for readers and writers

## *…::::Semaphores and Global variables same as APPROACH # 01:::…..*

## *Macros:*

- parm_reader - predefines the number of readers to be present
- parm_writer - predefines the number of writers to be present
- rrand_int - this the fixed time for which all readers perform a 'read'

## *Functions for semaphore:*

- **down() -** decrements the semaphore value.
- **up() -** increments the semaphore value.

## *Functions for times:*

- getnstimeofday() - inorder to get recent system time
- msleep_interruptible() - inorder to simulate the waiting

## *….:::::READER/WRITER functions perform the same task as APPROACH # 01:::…*

## *Problems Faced:*

- The writers can have variable times in which they perform as shown below

```
[ 4431.168441] Entered by Writer Thread 1 at 32:25 | Will take t = 3 sec to write
[ 4434.368785] Exit by Writer Thread 1 at 32:28

[ 4434.368813] Entered by Writer Thread 2 at 32:28 | Will take t = 4 sec to write
[ 4438.464067] Exit by Writer Thread 2 at 32:32

[ 4438.464097] Entered by Writer Thread 3 at 32:32 | Will take t = 3 sec to write
[ 4441.536105] Exit by Writer Thread 3 at 32:35

[ 4441.536150] Entered by Writer Thread 4 at 32:35 | Will take t = 1 sec to write
[ 4442.560391] Exit by Writer Thread 4 at 32:36

[ 4442.560422] Entered by Writer Thread 5 at 32:36 | Will take t = 3 sec to write
[ 4445.631272] Exit by Writer Thread 5 at 32:39

[ 4445.631303] Entered by Writer Thread 6 at 32:39 | Will take t = 2 sec to write
[ 4447.647690] Exit by Writer Thread 6 at 32:41
```

- rrand_int is defined as a macro instead of using the random time in the struct passed to the function. Hence readers perform as shown below

```
[ 4430.161945] Request by Reader Thread 1 at 32:24
[ 4430.161945] Entered by Reader Thread 1 at 32:24 | Will take t = 1 sec to read

[ 4430.161955] Request by Reader Thread 2 at 32:24
[ 4430.161956] Entered by Reader Thread 2 at 32:24 | Will take t = 1 sec to read

[ 4430.161980] Request by Reader Thread 3 at 32:24
[ 4430.161980] Entered by Reader Thread 3 at 32:24 | Will take t = 1 sec to read

[ 4430.161989] Request by Reader Thread 4 at 32:24
[ 4430.161989] Entered by Reader Thread 4 at 32:24 | Will take t = 1 sec to read

[ 4430.161997] Request by Reader Thread 5 at 32:24
[ 4430.161997] Entered by Reader Thread 5 at 32:24 | Will take t = 1 sec to read

[ 4430.162011] Request by Reader Thread 6 at 32:24
[ 4430.162011] Entered by Reader Thread 6 at 32:24 | Will take t = 1 sec to read
```

- This is because the readers cannot have the same as for readers having variable times of execution, the kernel threads do not respond effectively causing a system overflow and hence ‘hangs’ the system.
- Moreover, the amount of readers and writers are fixed as allocation of random readers and writer at run time caused a lot of problems for kernel hence the simulation is for a fixed number of readers and writers however not necessarily equal.

## *System call linkage file code(.c):*

```
#include <linux/init.h>
#include <linux/semaphore.h>
#include<linux/module.h>
#include<linux/kernel.h>
```

```
#include<linux/kthread.h>
#include<linux/sched.h>
#include<linux/time.h>
#include<linux/timer.h>
#include<linux/delay.h>
#include<asm/delay.h>
#include<linux/random.h>

#define parm_reader 6
#define parm_writer 6
#define rrand_int 1000

struct read_info{
      int id;
      int time;
};

int writer(void *i);     // writer thread
int reader(void *i);      // reader threads
int i,j,k,l,m,n;
long int avgtime,avg_time;
int wrand_int=0;

static struct semaphore mutex;            // semaphore for writer entry
into cs
static struct semaphore rwmutex;          // semaphore for shared variable
read_count safety
static struct semaphore avgmutex;         // semaphore for shared variable
avg_time safety
int read_count=0;       // number of readers inside cs

int writer(void * param){


      int id = *(int*)param;
      struct timespec rs;
      struct timespec es;
      struct timespec ls;


      printk("\n");
      // calculating request time
      getnstimeofday(&rs);
      printk("Request by Writer Thread %d at
%.2lu:%.2lu\n",id,(rs.tv_sec/60)%60,rs.tv_sec%60);

      down(&rwmutex); // wait writer

      // calculating entry time

      get_random_bytes(&wrand_int,sizeof (wrand_int));
      if(wrand_int<0)
      wrand_int = -1*wrand_int;
      wrand_int = ( wrand_int % 4 ) + 1;
```

```
	wrand_int*=1000;

	getnstimeofday(&es);

	printk("Entered by Writer Thread %d at %.2lu:%.2lu | Will take t =
%d sec to write\n",id,(es.tv_sec/60)%60,es.tv_sec%60,wrand_int/1000);

	msleep_interruptible(wrand_int);

	getnstimeofday(&ls);
	printk("Exit by Writer Thread %d at
%.2lu:%.2lu\n",id,(ls.tv_sec/60)%60,ls.tv_sec%60);

	up(&rwmutex);	// signal writer

	down(&avgmutex);
	// adding waiting time to shared variable avg_time
	avg_time += es.tv_sec-rs.tv_sec;
	up(&avgmutex);

	printk("\n");
	return 0;
}

int reader(void * param)
{

	struct read_info *ptr = (struct read_info*)param;
	struct timespec rs;
	struct timespec es;
	struct timespec ls;

	printk("\n");
	getnstimeofday(&rs);
	printk("Request by Reader Thread %d at %.2lu:%.2lu\n",ptr-
>id,(rs.tv_sec/60)%60,rs.tv_sec%60);


	down(&mutex);	// wait for read_count access permission
	read_count++;		// increment read count as new reader is
entering
	if(read_count==1)	// only if this is the first reader
		down(&rwmutex); // then wait for cs permission
	up(&mutex);	// signal mutex

	getnstimeofday(&es);
	printk("Entered by Reader Thread %d at %.2lu:%.2lu | Will take t = 1
sec to read\n",ptr->id,(es.tv_sec/60)%60,es.tv_sec%60);

	msleep_interruptible(rrand_int);

	down(&mutex);	// wait for read count access permission
	read_count--;		// decrement readcount as we are done reading
	if(read_count==0)	// only if this is the last reader
```

```
            up(&rwmutex); // signal rwmutex

        getnstimeofday(&ls);
        printk("Exit by Reader Thread %d at %.2lu:%.2lu\n",ptr-
>id,(ls.tv_sec/60)%60,ls.tv_sec%60);

        up(&mutex);

        down(&avgmutex);
        // adding waiting time to shared variable avg_time
        avg_time += es.tv_sec-rs.tv_sec;
        up(&avgmutex);

        printk("\n");
        return 0;
}

asmlinkage long sys_rwprob(void)
{

     // creating n pthreads
    static  struct task_struct *w[parm_writer],*r[parm_reader];
    int w_attr[parm_writer];
    struct read_info read[parm_reader];

    // initialising semaphores
    sema_init(&mutex,1);
    sema_init(&rwmutex,1);
    sema_init(&avgmutex,1);

      for (j = 0; j < parm_reader; ++j) {

            get_random_bytes(&read[j].time,sizeof (read[j].time) );
            if(read[j].time<0)
            read[j].time = -1*read[j].time;
            read[j].time = ( read[j].time % 4 ) + 1;
            read[j].time*=1000;

            read[j].id=j+1;

                r[j]=kthread_create(reader,&read[j],"reader");
            if(r[j])
            wake_up_process(r[j]);
    }

      for (i = 0; i < parm_writer; ++i) {
        w_attr[i]=i+1;
                w[i]=kthread_create(writer,&w_attr[i],"writer");
      if(w[i])
      wake_up_process(w[i]);
    }

      for(l=0;l<parm_reader;l++)
                kthread_stop(r[l]);
```

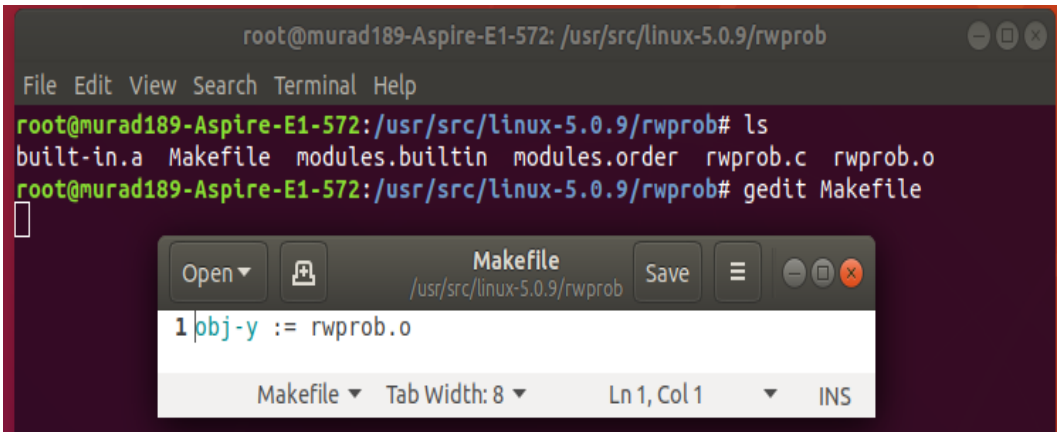
```
        for(k=0;k<parm_writer;k++)
                kthread_stop(w[k]);

        avgtime=((avg_time)/((parm_reader+parm_writer)));
        printk("\nAvg time spent by readers and writers in critical
section(sec): %ld\n",avgtime);


        return 0;
}
```

### *C Code to call the system call (code.c):*

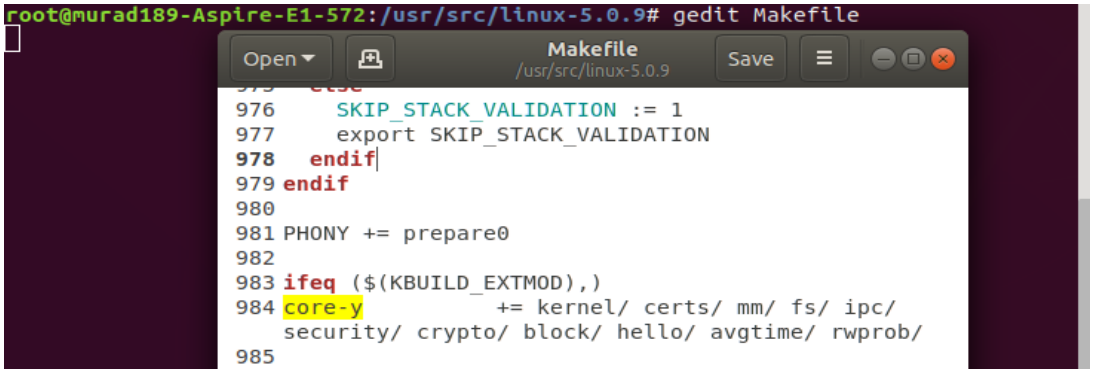
```
#include<stdio.h>
#include<unistd.h>
#include<sys/syscall.h>

int main(){
        long int s = syscall(337);
        printf("\n.:SYSTEM CALL VALUE:. %ld\n",s);
        return 0;
}
```

# *Screenshots:*

- This shows the Makefile for the system call stored in the c file 'rwprob'




- This shows that the system call for 'rwprob' is added to Makefile of the kernel



- This shows the linkage file for the system call

```
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9# cd include/linux
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/include/linux# gedit syscalls.h
```

syscalls.h
/usr/src/linux-5.0.9/include/linux

```
1314
1315        return old;
1316 }
1317
1318 asmlinkage long sys_hello(void);
1319
1320 asmlinkage long sys_avgtime(int,int,int);
1321
1322 asmlinkage long sys_rwprob(void);
1323
1324 #endif
```

- This shows that the system call has been added to the syscall_64 table

```
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/arch/x86/entry/syscalls# ls
Makefile  syscall_32.tbl  syscall_64.tbl  syscallhdr.sh  syscalltbl.sh
root@murad189-Aspire-E1-572:/usr/src/linux-5.0.9/arch/x86/entry/syscalls# gedit
syscall_64.tbl
```

syscall_64.tbl
/usr/src/linux-5.0.9/arch/x86/entry/syscalls

```
341 330     common  pkey_alloc          __x64_sys_pkey_alloc
342 331     common  pkey_free           __x64_sys_pkey_free
343 332     common  statx               __x64_sys_statx
344 333     common  io_pgetevents       __x64_sys_io_pgetevents
345 334     common  rseq                __x64_sys_rseq
346 335     64      hello               sys_hello
347 336     common  avgtime             __x64_sys_avgtime
348 337     64      rwprob              sys_rwprob
349
```

- This verifies the that the system call compiled successfully

```
murad189@murad189-Aspire-E1-572:~$ gcc -o a code.c
murad189@murad189-Aspire-E1-572:~$ ./a

.:SYSTEM CALL VALUE:. 0
murad189@murad189-Aspire-E1-572:~$
```

## *Output:*

```
[ 4430.159540] module1: module verification failed: signature and/or required key

[ 4430.161945] Request by Reader Thread 1 at 32:24
[ 4430.161945] Entered by Reader Thread 1 at 32:24 | Will take t = 1 sec to read

[ 4430.161955] Request by Reader Thread 2 at 32:24
[ 4430.161956] Entered by Reader Thread 2 at 32:24 | Will take t = 1 sec to read

[ 4430.161980] Request by Reader Thread 3 at 32:24
[ 4430.161980] Entered by Reader Thread 3 at 32:24 | Will take t = 1 sec to read

[ 4430.161989] Request by Reader Thread 4 at 32:24
[ 4430.161989] Entered by Reader Thread 4 at 32:24 | Will take t = 1 sec to read

[ 4430.161997] Request by Reader Thread 5 at 32:24
[ 4430.161997] Entered by Reader Thread 5 at 32:24 | Will take t = 1 sec to read

[ 4430.162011] Request by Reader Thread 6 at 32:24
[ 4430.162011] Entered by Reader Thread 6 at 32:24 | Will take t = 1 sec to read

[ 4430.162019] Request by Writer Thread 1 at 32:24

[ 4430.162027] Request by Writer Thread 2 at 32:24

[ 4430.162034] Request by Writer Thread 3 at 32:24

[ 4430.162042] Request by Writer Thread 4 at 32:24

[ 4430.162050] Request by Writer Thread 5 at 32:24

[ 4430.162053] Request by Writer Thread 6 at 32:24
[ 4431.168378] Exit by Reader Thread 1 at 32:25

[ 4431.168402] Exit by Reader Thread 6 at 32:25

[ 4431.168407] Exit by Reader Thread 5 at 32:25

[ 4431.168413] Exit by Reader Thread 4 at 32:25

[ 4431.168418] Exit by Reader Thread 3 at 32:25

[ 4431.168422] Exit by Reader Thread 2 at 32:25

[ 4431.168441] Entered by Writer Thread 1 at 32:25 | Will take t = 3 sec to write
[ 4434.368785] Exit by Writer Thread 1 at 32:28

[ 4434.368813] Entered by Writer Thread 2 at 32:28 | Will take t = 4 sec to write
[ 4438.464067] Exit by Writer Thread 2 at 32:32

[ 4438.464097] Entered by Writer Thread 3 at 32:32 | Will take t = 3 sec to write
[ 4441.536105] Exit by Writer Thread 3 at 32:35

[ 4441.536150] Entered by Writer Thread 4 at 32:35 | Will take t = 1 sec to write
[ 4442.560391] Exit by Writer Thread 4 at 32:36

[ 4442.560422] Entered by Writer Thread 5 at 32:36 | Will take t = 3 sec to write
[ 4445.631272] Exit by Writer Thread 5 at 32:39

[ 4445.631303] Entered by Writer Thread 6 at 32:39 | Will take t = 2 sec to write
[ 4447.647690] Exit by Writer Thread 6 at 32:41

[ 4447.647717]
               Avg time spent by readers and writers in critical section(sec): 4
murad189@ubuntu:~$
```