National University of Computer & Emerging Sciences

Karachi Campus

Project: System call for Spin Lock

GROUP MEMBERS:

MUHAMMAD HATIF MUJAHID(20K-0218)

WAHAJ JAVED ALAM (20K-0208)

MINHAL IRFAN(20K-0316)

Date of Submission: 26th May,2022

Introduction:

Spin locks are a low-level synchronization mechanism suitable primarily for use on shared memory multiprocessors. When the calling thread requests a spin lock that is already held by another thread, the second thread spins in a loop to test if the lock has become available. When the lock is obtained, it should be held only for a short time, as the spinning wastes processor cycles. Callers should unlock spin locks before calling sleep operations to enable other threads to obtain the lock.

It has only two values: 'locked' and 'unlocked'. If the process is in its critical section and lock is available then it sets the locked bit and acquires the lock . If the lock is used by somebody then it goes into a tight section and repeatedly checks until the lock is available. It is also called busy waiting .

Apart from that, SpinLocks are often used when the critical section is very small, small as in very less access to shared data structures.

Methodology:

When the calling thread requests a spin lock that is already held by another thread, the second thread spins in a loop to test if the lock has become available. When the lock is obtained, it should be held only for a short time, as the spinning wastes processor cycles. Callers should unlock spin locks before calling sleep operations to enable other threads to obtain the lock.

When performing any lock, a trade-off is made between the processor resources consumed while setting up to block the thread and the processor resources consumed by the thread while it is blocked. Spin locks require few resources to set up the blocking of a thread and then do a simple loop, repeating the atomic locking operation until the lock is available. The thread continues to consume processor resources while it is waiting.

For this project we used a dynamic spinlock which is initialized at runtime. Below is the description of some of the functions and a macro that we used:

1. **SYS_DEFINE1**

   This Macro allows us to define a system call by passing one parameter from the function caller. It takes the following parameters: The first one is the name of the

system call which we are making at the moment which in our case was spinLock as mentioned in the header file syscalls.h. The second parameter is the data type of the variable which we will be receiving and the third parameter is a placeholder to represent the variable. All the parameters are comma separated. This is done at the kernel level whereas the system call invoking syntax is the same as syscall(355,&variable); we have mentioned &variable here since we are passing the variable by reference.

2. **spin_lock_init()**

   The spin_lock_init() function is intended to be used when you need to initialize a spinlock at run time.

3. **spin_lock_irqsave()**

   The spin_lock_irq* functions are important if you expect that the spinlock could be held in an interrupt context. Reason being that if the spinlock is held by the local CPU and then the local CPU services an interrupt, which also attempts to lock the spinlock, then you have a **deadlock**. So to prevent a deadlock we save the interrupt state.

4. **spin_unlock_irqrestore()**

   The spin_unlock_irqrestore* basically unlocks the spinlock and restores the interrupt flag which was saved. Both spin_lock_irqsave and spin_lock_irqrestore work in tandem with each other.

5. **pr_info()**

   Pr_info is included in the printk header but is used to display kernel info level messages as well.

6. **printk()**

   A similar printf function to print kernel related messages in the kernel logs. Its output is shown in the kernel using the dmesg command.

In the end, the main feature of our function call is that it will accept a variable from any .c program and decrement using spinlocks. This ensures that no synchronization and/or reader-writer problem occurs.

Steps of implementation:

1. Go to the Directory arch/x86/entry/syscalls/ and open the file syscall_64.tbl and add the following line:

   335 common spinLock sys_spinLock

2. Now go to the Directory include/linux/syscalls.h and add the following line:

   Asmlinkage long sys_spinLock(int *x);

3. Now go to kernel/Makefile and add

   spinLock.o to the list of objects

4. Finally go to kernel and create a file spinLock.c and add the following snapshot of code:

```
#include<linux/init.h>
#include<linux/interrupt.h>
#include<linux/kernel.h>
#include<linux/module.h>
#include<linux/spinlock.h>
#include<linux/syscalls.h>


SYSCALL_DEFINE1(spinLock,int*,x)
{
static spinlock_t dynamic;
unsigned long flags;
spin_lock_init(&dynamic);
spin_lock_irqsave(&dynamic,flags);
pr_info("Locked dynamic spinlock\n");
*x = *x - 1;
spin_unlock_irqrestore(&dynamic,flags);
pr_info("Unlocked dynamic spinlock\n");
printk("New value: %d\n",*x);
return 0;
}
```

5. Finally compile the kernel and install modules and restart.
6. The system call is now successfully added.

Final Output:

1. Go to Desktop.
2. Add a file test.c
3. Write the code and compile
    a. gcc -o t test.c
4. Now, run the file to see the output displayed on the terminal.
5. Also, write dmesg on console to see the value after spinlock is used as well.

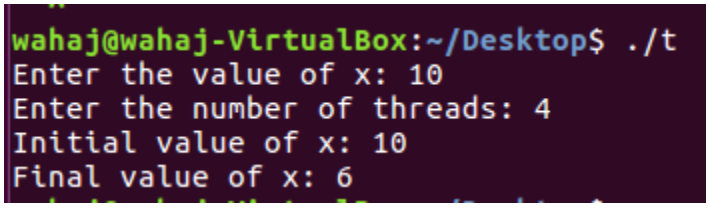
Driver Code:

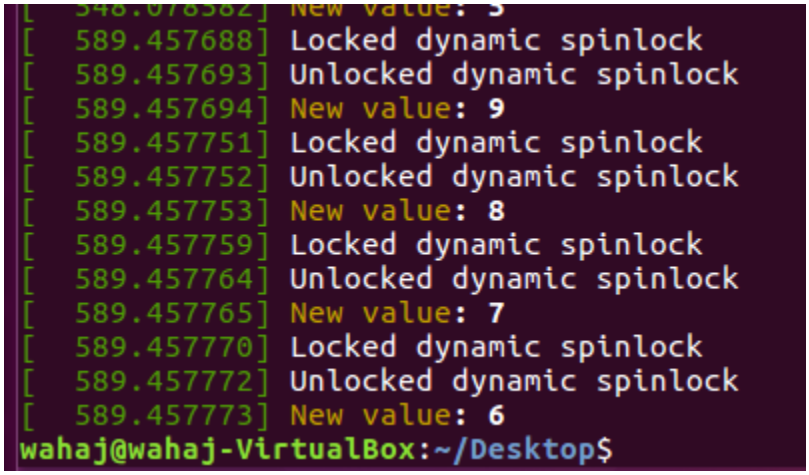
```
#include<sys/syscall.h>
#include<linux/kernel.h>
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

int n,x;

void *handler(){
        syscall(335,&x);
return 0;
}
int main(){
        printf("Enter the value of x: ");
        scanf("%d",&x);
        printf("Enter the number of threads: ");
        scanf("%d",&n);
        printf("Initial value of x: %d\n",x);
        pthread_t threads[n];
        for(int i=0;i<n;i++){
                pthread_create(&threads[i],NULL,handler,NULL);
        }

        for(int i=0;i<n;i++){
                pthread_join(threads[i],NULL);
        }
        printf("Final value of x: %d\n",x);
return 0;
}
```

Explanation:

The code asks for the number of pthreads and the variable x which is supposed to be decremented.

Output Screenshots:





References:

[Using Spin Locks - Multithreaded Programming Guide]

[What is a simple description of a spinlock? - Quora]

[Difference between Spinlock and Semaphore - GeeksforGeeks]

[Adding a Hello World System Call to Linux Kernel | by Anubhav Shrimal]

[How to pass parameters to Linux system call? - Stack Overflow]