

System calls through Semaphore

OS/ Section: D

Ahsun asim K163764

muzammil asrar khan k163769

**Background**

A semaphore is a variable or abstract data type that is used for controlling access, by multiple processes, to a common resource in a concurrent system such as a multiprogramming operating system. Semaphore is very helpful when we have multiple processes running through a critical section of code.

**Project Objective**

The built-in functions of semaphore cannot be directly accessed in system calls, we will implement the concept in our own way by making two system call of post and wait. Through this post and wait system call we solved the bounded buffer problem(as an example).

**Methodology**

We used linux 3.0.1 for implementation of this project, because the new version crashed twice, so we had to go with an older version. Firstly we added our system calls in our system call table which was located in root =>/arch/x86/syscalls/ and the table was stored in file syscall\_32.tbl. Entry number for wait and post system call was 353 and 354 respectively. Then we made separate directories for wait and post in /root/linux. The code of each system call was saved in their respective directory along with their Makefile. Then we edited the predefined Makefile in /root/linux by adding /wait /post to core-y. Finally we make, make install and then reboot. Afterword we implemented our system call in a test code for bounded buffer problem.

**Code for wait system call:**

#include <linux/kernel.h>

#include <linux/syscalls.h>

#include <asm/unistd.h>

asmlinkage long sys\_wait(int s)

{

static int n=1;

printk("Thread %d entering wait.\n",n);

n=n+1;

printk("Busy wait. \n");

s=s-1;

return s;

}

**Code for post system call:**

#include <linux/kernel.h>

#include <linux/syscalls.h>

asmlinkage long sys\_post(int x)

{

static int m=1;

printk("Thread %d entering post. \n",m);

m=m+1;

x=x+1;

return x;

}

**Code for Implementation:**

#include <linux/unistd.h>

#include <sys/syscall.h>

#include <stdio.h>

#include <linux/kernel.h>

#include <sys/types.h>

#define \_\_NR\_wait 353

#define \_\_NR\_post 354

const int n=5;

char p[10] = "Producer:";

char c[10] = "Consumer:";

int count = 1;

int count1 =1;

int mutex=1; //binary semaphore.

int full = 0; //counting semaphore.

int empty = n; //counting semaphore.

long int signal;

long int wait=0;

void producer(){

printf("%s\n",p);

if(empty!=0 && empty > 0){

while(empty!=0 && empty > 0){

wait = syscall(353, empty);

empty=wait;

wait = syscall(353, mutex);

mutex=wait;

printf("producer %d added to the buffer\n", count);

count++;

sleep(1);

signal = syscall(354, mutex);

mutex=signal;

signal = syscall(354, full);

full=signal;

}

}

else

printf("ERROR Full buffer.\n");

}

void consumer()

{

printf("%s\n",c);

if(full!=0 && full > 0){

while(full!=0 && full > 0)

{

wait = syscall(353, full);

full=wait;

wait = syscall(353, mutex);

mutex=wait;

printf("Item %d removed from the buffer\n",count1);

count1++;

sleep(1);

signal = syscall(354, mutex);

mutex= signal;

signal = syscall(354, empty);

empty= signal;

}

}

else

printf("ERROR Empty buffer.\n");

}

int main()

{

consumer();

producer();

producer();

consumer();

producer();

consumer();

producer();

consumer();

producer();

producer();

consumer();

producer();

producer();

consumer();

producer();

return 0;

}

**Problems Faced**

We faced problems while running the make command. After more than 15 attempts we finally managed to successfully fulfill this task. The main hurdle was to include 2 system calls together. The system crashed twice with a fatal error in kernel.

References

<https://www.youtube.com/watch?v=5rr_VoQCOgE>

<https://tssurya.wordpress.com/2014/08/19/adding-a-hello-world-system-call-to-linux-kernel-3-16-0/>

Lab manual on semaphore