Barrier Synchronization Project Report
Kernel Programming - CS401

Prasanth P
143050059

1 Problem Scope and Objectives

The objective of the project is to create a Barrier Synchronization mechanism in Linux Kernel.

There should be an initialization system call `initBarrier(gid, nproc)` by which a **barrier group** with id 'gid' is initialized with **the number of processes** 'nproc'. That is, **nproc** number of processes are supposed to block at the barrier **gid**.

There should also be a `block(gid)` system call which when called, the caller will block on the barrier with id 'gid'. When all the 'nproc' processes reach the barrier (calls `block()`), all those processes are unblocked and allowed to continue together.

2 Methodology and Approach

To implement the barrier synchronization mechanism, a new system call is added to the kernel. The system call takes 3 arguments — opcode, arg1, and arg2. It passes these arguments and the `current` task_struct pointer into a hook function.

When a program calls the `initBarrier(gid, nproc)` function, it is translated to the system call by a header file. The hook function tries to create a group with given group id (gid) and number of processes (nproc) and adds it to a linked list of group elements.

When block(gid) function is called by a process, the header file translates it into the system call which then blocks the process by sending SIGSTOP to it and adds it to the tasks-list of the group element with the given gid. It also decrements the nproc of the group. If nproc is 0 now (this was the last process of the group), all the processes in the group are unblocked.

3 Components

3.1 System Call: sys_pbarrier

The system call takes 3 arguments
1. operation
2. arg1
3. arg2

It then obtains the task_struct pointer of the calling process using macro 'current'. A hook function is called and this task_struct pointer along with the 3 arguments given above are passed into it.

3.2 LKM: barrier_sys.c

The loadable kernel module implements the hook function `pbarrier`. It also initializes a linked list of barrier groups and a sysfs interface for providing current stats. The parts of the lkm are given below.

Linked List

A linked list of barrier groups is implemented in '`grptskds.c`'. A barrier-element in the linked list has 3 members
1. gid : Group ID
2. nproc: No. of processes for the barrier group
3. tsks : A linked list of **task_struct pointers** — tasks blocked on the barrier

Various functionalities possible in the linked list are:

1. `createGroup(gid, nproc)`: creates a new group with given gid and nproc, and adds it to the list. It then returns the address of that group element. Returns NULL if that gid is already taken.
2. `getGroup(gid)`: returns the linked list element (group) with the given gid. Returns NULL if not found.
3. `insertTask(gid, tsk)`: inserts the task_struct 'tsk' into the group-node with id 'gid'. As the task is inserted, the counter 'nproc' is decremented. Returns the following values
   (a) 0: task inserted and nproc decremented successfully.
   (b) 1: nproc reached zero, denoting it was the last process for the group. All processes in the group must be freed.
   (c) 2: group id not present.
   (d) 3: task already present in the group.
   (e) 4: invalid (negative) gid.
4. `printGroups(gid, buf)`: if gid is -1, state of all groups is entered into the buffer 'buf'. Otherwise, state of that group with given gid is copied into 'buf'.
5. `freeGroup(gid)`: Every process blocked in the barrier group 'gid' is allowed to continue and the 'tsks' list is freed. The group element in the linked list is also freed. The given gid is free to be used again.

6. `freeLists()`: All the groups' task_struct list tsks is freed, and the linked list of group-elements are also safely freed.

pbarrier function

The function takes 4 arguments: the `task_struct` of the process to be blocked, an `operation code`, and two arguments `arg1` and `arg2`. A spin lock is used for mutual exclusion. An appropriate action is taken according to the operation code.
If the operation code is 0 (BARRIER_INIT):
arg1 is considered as gid and arg2 as nproc. `createGroup(arg1,arg2)` is called and a new group is created with given gid and nproc.
If the operation code is 1 (BARRIER_BLOCK):
arg1 is taken as gid and arg2 is ignored. `insertTask(task, arg1)` is called and the task is added to the task-list of group with given 'gid'. If the function returns 0, the process is blocked by sending **SIGSTOP** . If the function returns 1, all the tasks blocked in that gid is unblocked and the lists freed by calling `freeGroup(arg1)`.

sysfs interface

Two sysfs files have been created in `/sys/barrier/`: `gid` and `stats`. gid is initialized to -1.
Inorder to obtain stats of all the existing groups, set gid to -1 (by writing into `gid` file) and read `stats` file.

```
echo "-1" > /sys/barrier/gid
cat /sys/barrier/stats
```

To obtain stats of group with id 'gid', write that into `gid` file and read the `stats` file.

```
echo {gid} > /sys/barrier/gid
cat /sys/barrier/stats
```

3.3 The Header File: pbarrier.h

A header file **pbarrier.h** was created which has wrapper functions `initGroup(gid, nproc)` and `block(gid)`, which internally calls the system call '`pbarrier`'.

initGroup(gid, nproc)

Initiates a new barrier group with group-id `gid` and number of processes `nproc`. A call with an existing group id will be neglected. When all the 'nproc' processes in the group reaches and blocks at the barrier, the processes will be allowed to continue, the group will be cleared, and the group id 'gid' will be free for a new group.

block(gid)

Blocks the caller process at the barrier of group 'gid'. If it was the last process out of the 'nproc' processes of that barrier, all the processes blocked at 'gid' barrier group will be released and they continue execution.

4 Evaluation

A c-program `tester` was created which takes command line args. The program prints from 0 to 9. After printing 4, the program blocks on the barrier `gid` by calling `block(gid)`. Only after the barrier block is unlocked, the remaining numbers are printed.
Usage:

```
./tester <gid> <nproc>
```

This will initialize a new process with the given gid and nproc by calling `initBarrier(gid, nproc)`. Then, the process prints from 0 to 4 and then call `block(gid)`.

```
./tester <gid>
```

The program prints from 0 to 4 and call `block(gid)` to block on the barrier.

In order to test the barrier synchronization, the program can be started with a nproc of 5. 5 instances of the program are executed with the **same gid**. Just when the 5th program reaches `block(gid)` system call, all of them should unblock and print from 5 to 9.

The state of the groups can be checked from the sysfs interface `/sys/barrier/stats`

5 Conclusions

The system call and the kernel module hook worked successfully. The evaluation by using the tester program gave expected results. The sysfs interface provided accurate (expected) stats while testing. The barrier synchronization worked correctly.