CS 111 – Lab2 Design Problem: Change Notification

Name: Huang, Baixiao
UID: 504-313-981
Name: Lu, Jiayi Lu
UID: 004-079-546

Description of Features:

Description & Option:

- option -n is added. So when it is specified, process with this option can watch different data region on the disk and get notified if data is changed.

Examples:

- A process is ran with:
  $ ./osprdaccess -n 0:10,13:14,100:200
  can subscribe to get notified when data of index from 0 to 9, 13 and from 100 to 199 (inclusive) are changed.

Implementation:

- New flag OSPRDIOCGETNOTIFIED for ioctl() is added in osprd.h.

- New option "-n" can be recognized with some modification in osprdaccess.c file. So when -n is read, main will call ioctl(devfd, OSPRDIOCGETNOTIFIED, notification_argument) Where notification_argument contains argument for -n option

```
 //Detect notification option
 if (argc >= 2 && strcmp(argv[1], "-n") == 0) {
     notification = 1;
     argv++, argc--;
     if (argc >=2) {
         notification_argument = argv[1];
         argv++;
         argc--;
     }
     goto flag;
  }
  if (notification && ioctl(devfd, OSPRDIOCGETNOTIFIED,
notification_argument) == -1){
     perror("ioctl OSPRDIOCGETNOTIFIED");
     exit(1);
}
```

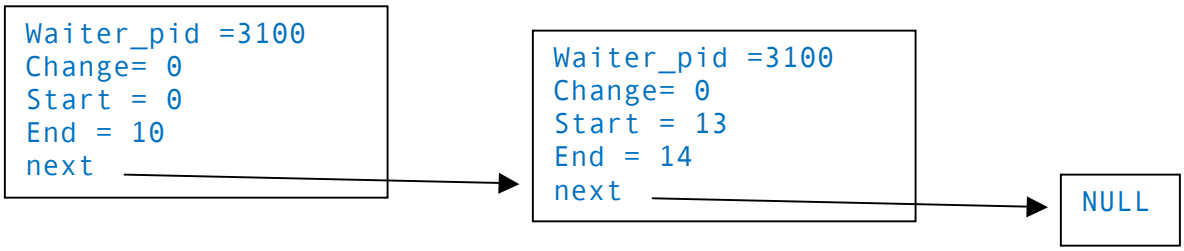
- A parser function parseNotifiArg() is added in osprd.c. Function can parse the argument like 0:10,13:14,100:200 into a notification list data structure.
```
char* parseNotifiArg(char* arg, int *start_ptr, int *end_ptr);
```

Data Structures:

```
notification_list{
    pid_t waiter_pid;
    int change;
    int start;
    int end;
    struct notification_list *next};
```

So every subscription for notification will be stored as a linked-list of notification_list structure.

For example, a process of pid 3100 subscribe to notification 0:10,13:14 would create a link list of:



This link list is stored in osprd_info structure:
```
    struct notification_list* notifi_list;
    struct notification_list* notifi_list_tail;
```

How Does Process Get Notified

We already know that process can subscribed to notification by parsing the argument into a link list stored in osprd_info, but how does process know something changed?

So we introduced checkNotification function in osprd_process_request. Before a process attempt to write to the disk, checkNotification function will be called. This function basically go through the notification link list and see if subscribed data region overlap with new data. If so, check if new data is different from the old data in subscribed data region. If the new data is different from the old data in subscribed region, it sets the change flag to 1.

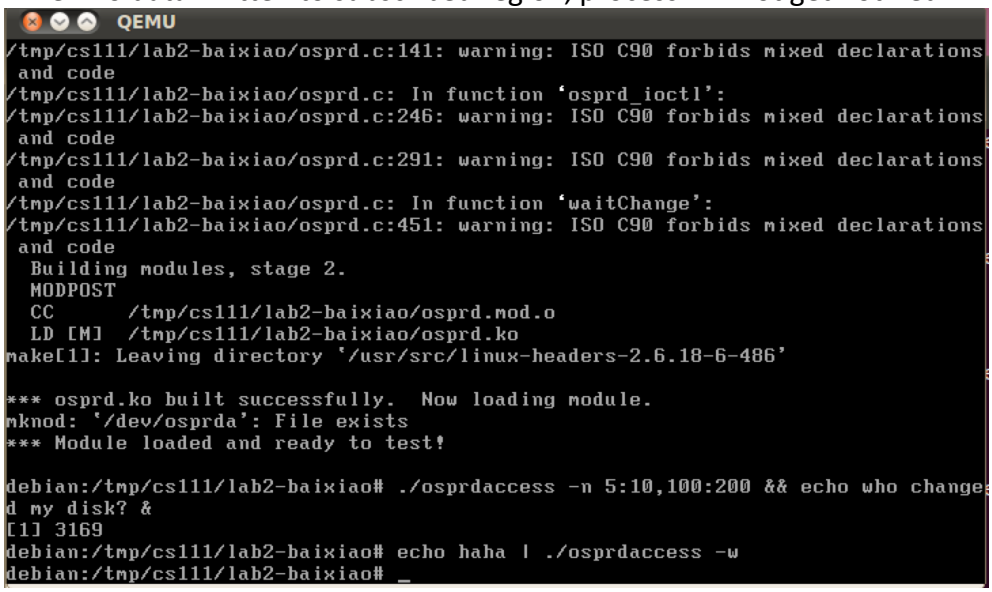
In this way, the process that subscribes to notification can constantly check to see if the change flag in the notification list is changed to 1. If so, process can unsubscribe from waiting for notification, and the node in the list will be deleted. Waiting functionality is implemented with wait_event_interruptible().

More details in osprd_ioctl():

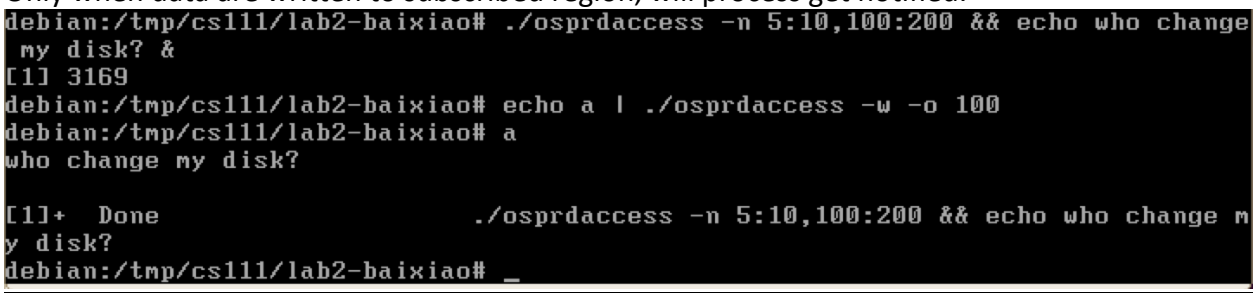
```
else if (cmd == OSPRDIOCGETNOTIFIED){
        int start=0, end=0;
        char* argument = (char*) arg;
        while (*argument!='\0') {
            struct notification_list * new_node = (struct
notification_list*)kmalloc(sizeof(struct notification_list),
GFP_ATOMIC);
            argument = parseNotifiArg(argument, &start, &end);
            new_node->change = 0;
            new_node->start = start;
            new_node->end = end;
            new_node->next = NULL;
            new_node->waiter_pid = current->pid;
            osp_spin_lock(&(d->mutex));
            if (d->notifi_list==NULL) {
                d->notifi_list = new_node;
                d->notifi_list_tail = new_node;
            }else{
                d->notifi_list_tail->next = new_node;
                d->notifi_list_tail = new_node;
            }
            osp_spin_unlock(&(d->mutex));
        }
        if (wait_event_interruptible(d->blockq, waitChange(current-
>pid,d))==-ERESTARTSYS) {
            return -ERESTARTSYS;
        }
        wake_up_all(&(d->blockq));
        r=0;
    }
```

Demo in QEMU:

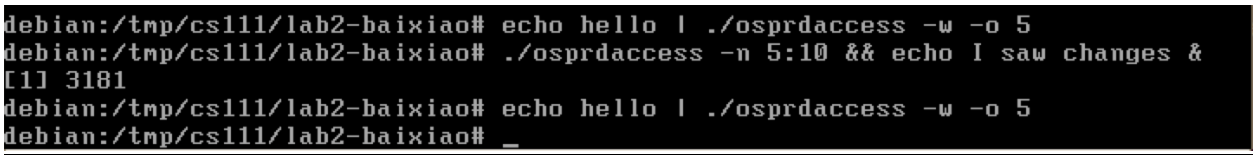
When no data written to subscribed region, process will not get notified.



Only when data are written to subscribed region, will process get notified.



When we write the same data to subscribed region, process will not get notified.



Summary:

This implementation of Notification is successful, and user is free to subscribe for multiple regions on disk for notification. And argument grammar is simple and elegant for users. Baixiao was responsible for coding, while Jiayi was responsible for debugging.