CS 342: OPERATING SYSTEM

Project 2: USER PROGRAMS

Submitted By:

Pradeep Bansal

10010143

Assignment 2d: Implement the rest of the system calls given in 3.3.4

CHANGED FILES

1. threads/thread.c
2. threads/thread.h
3. userprog/process.c
4. userprog/process.h
5. userprog/syscall.c

DATA STRUCTURES:

1. In syscall.c

```
struct fd_elem
 {
   int fd; /* the unique file descriptor number returns to user process. */
   struct file *file;/*file associated*/
   struct list_elem elem;/*list of elements*/
   struct list_elem thread_elem;
 };
```

ALGORITHMS

Read:

static int sys_read (int fd, void *buffer, unsigned size);

First, check if the buffer and buffer + size are both valid pointers, if not, exit(-1). Acquire the fs_lock (file system lock). After the current thread becomes the lock holder, check if fd is in the two special cases: STDOUT_FILENO and STDIN_FILENO. If it is STDOUT_FILENO, then it is standard output, so release the lock and return -1. If fd is STDIN_FILENO, then retrieve keys from standard input. After that, release the lock and return 0. Otherwise, find the open file according to fd number from the open_files list. Then use file_read in filesys to read the file, get status. Release the lock and return the status.

Write:

static int sys_write (int fd, const void *buffer, unsigned length);

Similar with read system call, first we need to make sure the given buffer pointer is valid. Acquire the fs_lock. When the given fd is STDIN_FILENO, then release the lock and return -1. When fd is STDOUT_FILENO, then use putbuf to print the content of buffer to the console. Other than these two cases, find the open file through fd number. Use file_write to write buffer to the file and get the status. Release the lock and return the status.

```
typedef int pid_t;
typedef int (*handler) (uint32_t, uint32_t, uint32_t);
static handler syscall_vec[128];
static struct lock file_lock;
```

Various Function names

1. static int sys_halt (void);
just power off .

2. static int sys_create (const char *file, unsigned initial_size);
Check file name if valid then call filesys_create (file, initial_size);

3. static int sys_open (const char *file);
Check validity of file and open it using filesys_open (file);
If this file name is not bad allocate fd and add it to open file list.

4. static int sys_close (int fd);
Find file using fd and close it and remove it from list of open file.

5. static int sys_exec (const char *cmd);
Check validity of argument and call process_execute (cmd);

6. static int sys_wait (pid_t pid);
Call process_wait (pid); explaine below.

7. static int sys_filesize (int fd);
Find file by fd and return file_length (f);

8. static int sys_tell (int fd);
Find file by fd and return file_tell (f);

9. static int sys_seek (int fd, unsigned pos);
Find file by fd and call file_seek (f, pos);

10. static int sys_remove (const char *file);
Check validity of file and call filesys_remove (file);

11.. static struct fd_elem *find_fd_elem_by_fd (int fd);
Scan file _list and return file whose fd is equal to argument.

12. static struct file *find_file_by_fd (int fd);
Return find_fd_elem_by_fd(fd);

13. static struct fd_elem *find_fd_elem_by_fd_in_process (int fd);
Scan current thread files and return which has same fd.

14. static int alloc_fid (void);
Allocate a unique fid.

wait-syscall is implemented in term of process_wait.(In Userprog/process.c)

- New struct child_status is defined to represent child's exit status. And a list of child_status is added into parent's thread struct, representing all children the parent owns. parent_id inside child's struct is also introduced to ensure child can find parent and set it's status if parent still exists.

- A child_status is created and added to list whenever a child is created, then parent will wait(cond_wait) if child has not already exited, child is responsible to set it's return status and wake up parent.

- Monitor is present in parent's struct to avoid race condition. Before checking or setting the status, Parent and Child both should acquire the monitor first. If parent is signaled or sees the child has exited, it will start to check the status.

- If child calls exit-syscall to exit, a boolean signal that indicate exit-syscall is called and the child's exit status will be set into the corresponding child_status struct in parent's children list.

- If child is terminated by kernel, the boolean signal mentioned above is remain as false, which will be seen by parent, and understood child is terminated by kernel.

- If parent terminates early, the list and all the structs in it will be free, then the child will find out the parent already exited and give up setting the status, continue to execute.

- Avoiding bad user memory access is done by checking before validating, checking mean using the function is_valid_ptr which checks whether it's NULL, whether it's a valid user address and whether it's been mapped in the process's page directory. Taking "write" system call as an example, the esp pointer and the three arguments pointer will be checked first, if anything is invalid, terminate the process. Then after enter into write function, the buffer beginning pointer and the buffer ending pointer(buffer + size - 1) will be checked before being used.

Second when error still happens, we handle it in page_fault exception.

Advantages of using File Descriptors:

1) Thread-struct's space is minimized
2) Kernel is aware of all the open files, which gains more flexibility to manipulate the opened files.

Disadvantages:
1) Consumes kernel space, user program may open lots of files to crash the kernel.
2) The inherits of open files opened by a parent require extra effort to be implement.