CSE 150 Project #2 Group #3

Group Members: Ryan Hules, Yiying Jie, Chi Hong Kou, Haoxian(Andy) Huang, Jonathan Huang, Nathan Huizar

Task I: File System

Goal: Implementing the file system calls - creat/e, open, read, write, close and unlink, documented in syscall.h. *Not* implementing a file system, but give user processes the ability to access a file system.

Two lines of code:

```
                openFile[0] = UserKernel.console.openForReading();
                openFile[1] = UserKernel.console.openForWriting();
```

must be included for stdin and stdout.

Global variable:

```
        protected OpenFile[] openFile;
```

Function implemented:

```
// This function is to search for an empty space in order to open the file. (Max 16 supported)
        private int searchSpace() {
                int fileDescriptor = -1;

                // support 16 files max;
                for (int i = 0; i < 16; i++)
                {
                        if(openFile[i] == null) {
                                fileDescriptor = i;
                                return fileDescriptor;
                        }
                }
                return -1;
        }
```

1. Creat/e
    a. Search within memory the input fileAddress
    b. See if the file already exists, that is, if the fileAddress is null
    c. If the file does not exist, create a new file
        i. Find the appropriate space empty space
        ii. Set boolean create to TRUE to create file
    d. Put in the fileDescriptor into file that was just created

Pseudocode:

```
openfile[fileDescriptor] =ThreadedKernel.fileSystem.open (filename, false);
if filename is null
        return -1; // create failed
// else…
int fileDescriptor = findEmpty();
if fileDescriptor != -1
        Create new file
else
        return -1; // create failed
```

2. Open
    a. Read the file from virtual memory according to user input fileAddress
    b. Have to make sure fileAddress is valid, implement check
    c. Open file, and make sure the value of create is FALSE
    d. Put in fileDescriptor

Code:

```
        private int handleOpen(int address) {

                // invalid address check;
                if (address < 0) {
                        return -1;
                }

                String file = readVirtualMemoryString(address, 256);

                // cannot open file does not exist.
                if (file == null) {
                        return -1;
                }

                // search for empty space;
                int fileDescriptor = searchSpace();

                /* if searchSpace returns -1, meaning it reached
                  16 max opening file. */
                if (fileDescriptor == -1) {
                        return -1;
                }

                else {
                        // the value of create should be false since we are only handling open
            right here;

                        OpenFile f = ThreadedKernel.fileSystem.open(file, false);

                        if(f == null) {
                            return -1;
                        }

                        else {

                            openFile[fileDescriptor] =
            ThreadedKernel.fileSystem.open(file, false);
                        }
                }
        }
```

3. Read
    a. Implement condition checks if greater than 16 or less than 0 or null
    b. Read file and save in temporary memory, return
    c. Write the content to the virtual memory address, return numbers that was successfully written

```
    private int handleRead(int fileDescriptor, int addr, int l) {
        if (fileDescriptor > 15 || fileDescriptor < 0) {
            return -1;
        }

        else if(openFile[fileDescriptor] == null) {
            return -1;
        }

        byte buffer[] = new byte[l];

        int readNum = openFile[fileDescriptor].read(buffer, 0, l);

        // couldn't read data;
        if(readNum <= 0) {
            return 0;
        }

        int writeNum = writeVirtualMemory(addr, buffer);
        return writeNum;
    }
```

4. Write
    a. Gives address of the write file, address, bytes of writing memory
    b. Implement condition checks if greater than 16 or less than 0 or open returns null/error (same as read)
    c. Read file into temporary memory, if error, return
    d. Write the contents to the disk, return
    e. Implement other condition checks, such as if successfully written data is greater than data to be stored, return

```
private int handleWrite(int fileDescriptor, int addr, int l) {
        // write data from virtual memory address into the file;

        // should not be greater than 15 or less than 0;
        if (fileDescriptor > 15 || fileDescriptor < 0) {
            return -1;
        }

        else if(openFile[fileDescriptor] == null) {
            return -1;
        }

        byte buffer[] = new byte[l];

        // store data into the temp buffer table;
        int readNum = readVirtualMemory(addr, buffer);

        if (readNum <= 0) {
            // no data read;
            return 0;
        }

        // now write the data in;
        int writeNum = openFile[fileDescriptor].write(buffer, 0, l);

        if (writeNum < l) {
            // error occured when writing, return error;
            return -1;
        }

        // return written;
        return writeNum;
    }
```

5. Close
    a. Implement condition checks, if greater than 16 or less than 0 or null
    b. Close open file, set the address to null

```
private int handleClose(int fileDescriptor) {

        // add comments later;

        // should not be greater than 15 or less than 0;
        if (fileDescriptor > 15 || fileDescriptor < 0) {
            return -1;
        }

        // or if the file does not exist, error;
        else if (openFile[fileDescriptor] == null) {
            return -1;
        }

        else {

            openFile[fileDescriptor].close();
            openFile[fileDescriptor] = null;

        }

        return 0;
    }
```

6. Unlink
    a. Remove data according to the address sent in
    b. Implement condition to check, if the file does not exist, no need to remove anything

Pseudocode:

```
if fileName is null
    return; // no need to unlink/delete
remove file, return;
```

Things to keep in mind: Close and Unlink goes together for the test case that was provided to us. Therefore, both must work/ with each other in order to perform the expected functionality.

Test Case ideas:

To test if handleFunction works, more cases should be considered. Such as testing each function individually and together. For example, there should a test case to test handleClose and handleUnlink separately, to see if both function functions the expected way.

Task II: Multiprocessing

Implement support for multiprogramming. The code given is restricted to running one user process at a time. We need to make it work for multiple user processes.

Read Virtual Memory

```
Int bytes= 0;
int n = 1024; //page size

        while (offset < data.length && length > 0) {
//Computes virtual page number
                int virPage = vaddr / n;
//Computes address offset
                int addressOffset = vaddr % n;
//Checks for out of bounds/invalid
                if (virPage < 0 || virPage >= pageTable.length) {
                break;
                }
//Find translation entry in page table using calculated pageNumber
TranslationEntry tran = pageTable[virPage];
if (!tran.valid)
Break;
//Set used bit true
tran.used = true;
//Store physical page number
int phyPage = tran.ppn;

int phyAddr = (phyPage * n) + addressOffset;
// remaining amount smallest from remaining;

int amount = Math.min(data.length - offset, Math.min(length, n - addressOffset));
//copy from memory to data using offset and physAddr as location
System.arraycopy(memory, phyAddr, data, offset, amount);
vaddr = vaddr + amount;
offset = offset + amount;
length -= amount;
bytes = bytes + amount; // compute remaining byte

return bytes;
```

Write Virtual Memory

```
Int bytes= 0;
int n = 1024; // page size
    while (offset < data.length && length > 0) {
//Compute virtual page num
        int virPage = vaddr / n;
//Compute offset
        int addressOffset = vaddr % n;
//Check for invalid/ out of bounds virtual page num
        if (virPage < 0 || virPage >= pageTable.length) {
        break;
        }
//Store entry gotten from pageTable at index of virtual page
TranslationEntry tran = pageTable[virPage];
//If bit not valid break;
if (!tran.valid)
Break;
//Set used bit to true
tran.used = true;
//Store physical page number from entry
int phyPage = tran.ppn;

int phyAddr = (phyPage * n) + addressOffset;
// remaining amount smallest from remaining;

int amount = Math.min(data.length - offset, Math.min(length, n - addressOffset));
//copy from memory to data using offset and physAddr as location
System.arraycopy(data, offset, memory, phyAddr, amount);
vaddr = vaddr + amount;
offset = offset + amount;
length -= amount;
bytes = bytes + amount;

return bytes;
```

- Add a variable to maintain list of free physical pages, as well as adding a free page into the list.
- Variable to remove first element of page list and return the number of free page.

- Modify **UserProcess.readVirtualMemory** and **UserProcess.writeVirtualMemory**, which copy data between the kernel and the user's virtual address space, to work with multiple user processes.
- Modify **UserProcess.loadSections()**so that it allocates the number of pages that it needs
    - This method should also set up the **pageTable** structure for the process so that the process is loaded into the correct physical memory pages.
- The physical memory of the MIPS machine is accessed through **Machine.processor().getMemory();** the total number of physical pages is **Machine.processor().getNumPhysPages()**.
- Implement **UserProcess.unloadSections()** to release page tables and back out physical pages

Test Case Ideas:

- **Test support of fragmentation in physical memory**
- **Test whether or not read and write works with an invalid range**
- **Test whether readVir and writeVir, read and wrote the right data**

Task III: System Calls

Exec

- Create new child process
- Execute program stored in specified file using said new process
- Return -1 on error

```
Public int exec(int address){
        String file = readVirtualMemoryString(address, 256);
        String[] arg = new String[argc];
        For (i: 1 through argc){
                Byte[] argAddr = new byte[4];
                if(readVirtualMemory(argv + i * 4, argAddr) > 0){
                        Arg[i] = readVirtualMemoryString(Lib.bytesToInt(argAddr, 0), 256);
                }
        }
        if(file is null || file doesn't end in .coff || file doesn't load || argc < 0 || argv doesn't fit in
        page space){
                Return -1;
        }
        UserProcess temp = UserProcess.newUserProcess();
```

```
    if(temp doesn’t execute){
        Return -1;
    }
    Make temp a child of this process
    Return temp’s process ID
}
```

Join

- Set up process ID counter (assume it will not overflow)
- Add thread parameters so children know who their parents are
- isChild() helper function?
- Check to ensure that threads can only be joined by their respective parent
- On call, check exit status of specified child process
- Sleep caller until said child process exits
- Disown child process on exit

```
Public int join(int processID, int status){
    if(this is current process){
        sleep()
    }
    if(status == -1){
        Return 0;
    }
    while (given process is a child){
        Disable interrupts
        If (status == 0 or 1){
            Disown child
            if(status == 0)
                Return 1;
            Else
                Return 0;
        }
        Else{

            Sleep
            Restore interrupts
        }
    }
    Return -1;
}
```

Exit

- Close open files
- Free Memory
- Disown any children
- Pass exit status to parent
- If last process, call Kernel.kernel.terminate()
- Terminate thread

```
Public void exit(int status){
        Close the process
        for(i: file array){
                Close file[i]
        }
        //disown children
        for(i: children){
                i.removeParent();
                children.remove(i);
        }
        if(last process){
                Machine.terminate();
        }
}
```

Task IV: Lottery Scheduler

- Priority donation
- Extend priority scheduler
- Waiting thread must transfer tickets
- Do not use array
- increase/Decrease should +1 and -1
- Min and Max values have changed

```
Public lotteryScheduler {

Public static final int priorityDefault = 1;
Public static final int priorityMaximum = Integer.MAX_VALUE;;
Public static final int priorityMinimum = 1;


        //will have its own set priority func that will pass in the thread and its associated priority
        Public void setPriorirty( KThread, priorty){
```

```
    //check if the priority is less than priorityMax
    //check if the priority is greater than priorityMin

    }

    //make a priortyQueue for the lotteryQueue
    Protected class lotteryQueue extends PriorityQueue {

        //here we will get the transfer priority and randomize
        private final Random rand;
        boolean transferPriority;

    //we also make sure that we do this for the threads after the current one
    Public ThreadState pickNextThread(){

    int totalTickets = getEffectivePriority();

    //check to see if tickets are greater than zero
    int winningTicket = totalTickets > 0 ? rand.nextInt(totalTickets) : 0;

    for (final ThreadState thread : waitThread) ->

        Lib.assertTrue(thread instanceof LotteryThreadState);

        winningTicket -= getEffectivePriority();

        if (winningTicket <= 0) ->
            return thread;

    }

    return null;
}


                }

    //we make another get effective priority because it is calculated a different way
    public int getEffectivePriority() {

    //check the transfer priority
    if (!this.transferPriority) ->
```

```
            get priorityMinimum;

        else if (this.changedPriority) ->

            // find new effective priorities
            this.efficientPriority = priorityMinimum;

            for (final ThreadState = waitThread) ->

                Lib.assertTrue(cur instanceof LotteryThreadState);
                //update effective priority
                efficientPriority += getEffectivePriority();


            this.changedPriority = false;

        //definied in Priority Scheduler
        return efficientPriority;
}

//a lot of the previous code we constructed for Priority Scheduler will be used to simplify the
//functions we are making for our new Lottery Scheduler
```