Project 2 - Initial Design Document

Chen Lijie Fan Haoqiang Bi Ke

Contents

1 Our git Repository

`https://github.com/wjmzbmr/nachos`

2 Implementation of System calls for File System

2.1 A simple illustration

Since in class FileSystem, we only have method open() and remove(), which means we need to implement unlink on our own by keeping a counter for each file opened.

2.2 Correctness Invariants

2.3 Declaration

UserProcess

- A static member processCounter, keeps the number of each process.
- A final member maxBuf, which is the maximum buffer size per read.
- A member processId.
- An array fileList of OpenFile with size 16 to store the opened file.
- Modification in UserProcess(), which set file Descriptor 0 and 1 to stdin and stdout.
- Modification in handleHalt()
- New methods: handleCreate(), handleOpen(), handleRead(), handleWrite(), handleClose(), handleUnlink(). With specified functionality in the task.

UserKernel

- A class FileManager, which keeps a counter for each file and whether it should be unlinked.
- A static subclass of FileManager, FileRecord, with two fields: counter and unlinked.
- A HashMap map in FileManager, map the file's name to the FileRecord.
- A Lock mutex in FileManager, ensuring that only one process can access to it.
- method open(), create(), unlink() and close() in FileManager, which will change the information for each file.

2.4 Description

The pseudo code follows:

UserProcess

```
procedure UserProcess()
    Disable Interruption
    processId ← processCounter++
    fileList ← new OpenFile[16]
    fileList[0] ← UserKernel.console.openForReading()
    fileList[1] ← UserKernel.console.openForWriting()
    Restore Interruption
end procedure
```

```
procedure HANDLEHALT()
    if processId != 0 then
        return -1
    end if
    Machine.halt()
end procedure
```

```
procedure HANDLECREATE(ADR)
    file ← readVirtualMemoryString(adr,256)
    if file == null then
        return -1
    end if
    idx ← 0
    while idx < 16 AND fileList[idx] != null do
        idx++
    end while
    if idx == 16 then
        return -1
    end if
    if NOT UserKernel.FileManager.create(file) then
        return -1
    end if
    openFile ← UserKernel.fileSystem.open(file,true)
    if openFile == null then
        return -1
    end if
    fileList[idx] ← openFile
    return idx
end procedure
procedure HANDLEOPEN()
    file ← readVirtualMemoryString(adr,256)
    if file == null then
        return -1
    end if
    idx ← 0
    while idx < 16 AND fileList[idx] != null do
        idx++
    end while
    if idx == 16 then
        return -1
    end if
    if NOT UserKernel.FileManager.open(file) then
        return -1
    end if
    openFile ← UserKernel.fileSystem.open(file,false)
    if openFile == null then
        return -1
    end if
    fileList[idx] ← openFile
    return idx
end procedure
```

```
procedure handleRead(idx,adr,buf)
    if idx is invalid OR adr is valid OR fileList[idx] is null then
        return -1
    end if
    file ← fileList[idx]
    while buf > 0 do
        toRead ← min(buf,maxBuf)
        read toRead bytes from file, and write it to adr
        if if in above an error occur then
            return -1
        end if
        buf ← buf - toRead
    end while
end procedure
```

```
procedure handleWrite(idx,adr,buf)
    if idx is invalid OR adr is valid OR fileList[idx] is null then
        return -1
    end if
    file ← fileList[idx]
    while buf > 0 do
        toRead ← min(buf,maxBuf)
        read toRead bytes from adr, and write it to the file
        if if in above an error occur then
            return -1
        end if
        buf ← buf - toRead
    end while
end procedure
```

```
procedure handleClose(idx)
    if idx is invalid then
        return -1
    end if
    file ← fileList[idx]
    name ← file.getName()
    file.close()
    fileList[idx] ← null
    if UserKernel.FileManager.close(name) then
        return 0
    else
        return -1
    end if
end procedure
```

```
procedure HANDLEUNLINK(ADR)
    if adr is invalid then
        return -1
    end if
    file ← readVirtualMemoryString(adr,256)
    if file == null then
        return -1
    end if
    if UserKernel.FileManager.unlink(file) then
        return 0
    end if
    return -1
end procedure
```

UserKernel

In class FileManager

```
procedure FILERECORD()
    unlinked ← false
    counter ← 0
end procedure
procedure FILEMANAGER()
    map ← new HashMap<String,FileRecord>()
end procedure
procedure CREATE(FILE)
    mutex.acquire()
    if NOT map.containsKey(file) then
        record ← new FileRecord()
        record.counter++
        map.put(file,record)
        mutex.release()
        return true
    else
        record ← map.get(file)
        if record.unlinked then
            mutex.release()
            return false
        end if
        record.counter++
        mutex.release()
        return true
    end if
end procedure
```

```
procedure OPEN(FILE)
    mutex.acquire()
    if NOT map.containsKey(file) then
        mutex.release()
        return false
    else
        record ← map.get(file)
        if record.unlinked then
            mutex.release()
            return false
        end if
        record.counter++
        mutex.release()
        return true
    end if
end procedure
procedure CLOSE(FILE)
    mutex.acquire()
    if NOT map.containsKey(file) then
        mutex.release()
        return false
    else
        record ← map.get(file)
        record.counter−−
        if record.counter == 0 AND record.unlinked then
            UserKernel.fileSystem.remove(file)
            map.remove(file)
        end if
        mutex.release()
        return true
    end if
end procedure
procedure UNLINK(FILE)
    mutex.acquire()
    if NOT map.containsKey(file) then
        mutex.release()
        return false
    else
        record ← map.get(file)
        if record.counter == 0 then
            UserKernel.fileSystem.remove(file)
            map.remove(file)
        else
            record.unlinked ← true
        end if
        mutex.release()
        return true
    end if
end procedure
```

2.5 Description of Tests

3 Implementation of Support for Multiprogramming

3.1 A simple illustration

Use a double linked list to maintain the currently available pages. So that we can make use of them efficiently.

3.2 Correctness Invariants

3.3 Declaration

UserKernel

- A linked list of Integer avaPages, which stores the currently available pages.
- A lock pagesMutex, which prevents multiple process from using the avaPages.

UserProcess

- Modifications in readVirtualMemory() and writeVirtualMemory().
- Modifications in loadSections()
- Modifications in unloadSections()
- Modifications in the constructor of UserKernel, which initialize the list of avaPages.

3.4 Description

The pseudo code follows:

UserKernel

```
procedure UserKernel()
    pagesMutex ← new Lock()
    while avaPages.size() < numPhypages do
        avaPages.add(new page)
    end while
end procedure
```

UserProcess

```
procedure readVirtualMemory(vaddr,data,offset,length)
    if vaddr is not valid then
        return 0
    end if
    length ← min(length, numPages * pageSize - vaddr)
    total ← 0
    begin ← Machine.process.pageFromAddress(vaddr)
    end ← Machine.process.pageFromAddress(vaddr + length - 1)
    for page ← begin to end do
        if page is invalid then
            return total
        end if
        read the corresponding bytes in page to data[offset..]
        update total and offset
    end for
    return total
end procedure
procedure writeVirtualMemory(vaddr,data,offset,length)
    if vaddr is not valid then
        return 0
    end if
    length ← min(length, numPages * pageSize - vaddr)
    total ← 0
    begin ← Machine.process.pageFromAddress(vaddr)
    end ← Machine.process.pageFromAddress(vaddr + length - 1)
    for page ← begin to end do
        if page is invalid then
            return total
        end if
        write the corresponding bytes in data[offset..] to the page
        update total and offset
    end for
    return total
end procedure
procedure loadSections()
    UserKernel.pagesMutex.acquire()
    if the avaPages.size() < numPages then
        UserKernel.pagesMutex.release() return false
    end if
    pageTable ← new TranslationEntry[numPages]
    for i ← 0 to numPages - 1 do
        page ← avaPages.poll()
        pageTable[i] ← new TranslationEntry(i,page,true,false,false,false)
    end for
    UserKernel.pagesMutex.release()
    ...
    vpn ← section.getFirstVPN()+i;
    pageTable[vpn].readOnly ← section.isReadOnly()
    section.loadPage(i, vpn);
    ..
    return true
end procedure
```

```
procedure UNLOADSECTIONS()
    UserKernel.pagesMutex.acquire()
    add all pages in pageTable to avaPages
    UserKernel.pagesMutex.release()
    close all those files opened in fileList
end procedure
```

3.5 Description of Tests

4 Implementation of System calls exec, join and exit

4.1 Correctness Invariants

4.2 Declaration

-

4.3 Description

4.4 Description of Tests

5 Implementation of LotteryScheduler

5.1 A simple illustration

5.2 Correctness Invariants

5.3 Declaration

5.4 Description

5.5 Description of Tests