Project 2 - Initial Design Document

Chen Lijie Fan Haoqiang Bi Ke

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

- Bullet-proof those system call so that user error won't break the kernel.
- Halt can only be called by the root process.
- The functionality and behavior of those call should match the what have been documented in syscall.h
- A user process should be able to open 16 file concurrently.

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 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 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 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 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 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 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

Standard tests

Do some standard tests to make sure our implementation works:

- open a file, read something from it, then close it.

- create a file, write some to it, then close it.
- make two process open the same file, then unlink it, then both process close it.
- unlink a file, and try to open it before it finally closed.

Bullet-proof tests

Do some special case test to make sure our implementation bullet-proof those user-level mistakes.

- test methods by some invalid parameters.
- write, read, close, unlink some files that doesn't exist.

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

- writeVirtualMemory() and readVirtualMemory() should return number of bits written or read if an error occurs.
- The page corresponding to COFF sections should be read-only.
- The memory should be efficiently used, that is, whenever the number of total available pages exceeds the pages a user process needs, that requirement should be satisfied.

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

Do some standard tests to make sure our implementation works:

- call writeVirtualMemory() and readVirtualMemory(), check whether they has the corresponding functionality.
- Test loading and unloading a .coff file.

Bullet-proof tests

Do some special case test to make sure our implementation bullet-proof those user-level mistakes.

- Call those functional by some invalid parameters.
- Try to require a huge memory which exceeds the entire memory of the machine.

4 Implementation of System calls exec, join and exit

4.1 A simple illustration

In this architecture, we only have one thread for each process, which will simplify things a lot.

4.2 Correctness Invariants

- in exit call, the last caller should terminate the machine.
- in exit call, it clean up the space used by the process.
- in joint call, a process can only be joined to its children.
- process ID should be unique for every process.
- Bullet-proof those system call so that user-level mistakes won't break the kernel.

4.3 Declaration

UserProcess

- A static member processCounter(already defined in Task I).
- A member thread of type UThread, denoting the thread of the current process.
- A member parent of type UserProcess, denoting the parent of the current thread.
- A member childList of type List<UserProcess>, denoting the children threads of the current thread.
- A member mapExitStatus of type Map<Integer,Integer>, which map the children process Id to its exit Status. And a lock mapExitStatusLock, we prevent atomic access for this map.
- Modification in UserProcess().
- Methods handleJoin(),handleExec(),handleExit(), with corresponding functionality.

4.4 Description

```
procedure UserProcess()
    Disable interruption.
    processId ← processCounter.
    processCounter ++.
    Enable interruption.
end procedure
```

```
procedure handleExit(status)
    Disable interruption.
    unLoadSections()
    set the parent to null for every process in childList
    if parent != null then
        parent.mapExitStatus.put(processId,status)
    end if
    if processId == 0 then
        terminate the Kernel.
    else
        terminate current thread.
    end if
end procedure
```

```
procedure handleJoin(pid,addr)
    child ← the process in childList with processId pid
    if no such process in above then
        return -1
    end if
    if child.thread !=null then
        child.thread.join()
    end if
    child.parent ← null
    remove child from childList
    exitStatusLock.acquire()
    status ← mapExitStatus.get(child.processId)
    mapExitStatus.remove(child.processId)
    exitStatusLock.release()
    if status is an abnormally exit then
        return 0
    end if
    write status to addr
    return 1
end procedure
```

```
procedure HANDLEEXEC(ADDR,ARGC,ARGV)
    if addr or argc is not valid then
        return -1
    end if
    file ← readVirtualMemoryString(addr,256)
    if file == null OR file is not valid then
        return -1
    end if
    arguments ← parse argc argument for argv.
    if the above parsing failed then
        return -1
    end if
    child ← new UserProcess
    if child.execute(file,arguments) then
        child.parent ← this
        childList.add(child)
        return child.processId
    else
        return -1
    end if
end procedure
```

4.5 Description of Tests

- Let a process call exit.
- Let a process join to another one, check the perform order.
- Let a process join to a thread which is not its children, see whether an error occurs.
- Create many threads, see that if their id is unique.

5 Implementation of LotteryScheduler

5.1 A simple illustration

Indeed we don't need to change much details from our previous implementation of priority Scheduler.

We only need to change the update for the priority donation and the way to pick the next thread.

5.2 Correctness Invariants

- Those method should be atomic.
- The threads are chosen with a probability proportional to the total number of tickets holding by itself and all threads waiting on the resources it holds.

5.3 Declaration

5.3.1 LotteryScheduler

- Implementation of getPriority(), setPriority() and getEffectivePriority().

5.3.2 Kthread

- Notice that the original Kthread Object has a member schedulingState, which can be used to record its scheduling state.

5.3.3 PriorityThreadQueue

- Make a subclass of ThreadQueue named LotteryThreadQueue. Which is supposed to maintain the threads waiting for this resource.
- A member variable resourceHolder, which points to the thread which holds the resource.
- A member variable sumPriority, which denoting the sum effective priority in the waiterQueue, set as 0 if waiters is empty.
- A binary search tree of SchedulingState waiters contains all the waiting threads.
- implementation of nextThread(), acquire(), waitForAccess().

5.3.4 SchedulingState

- member variables thread, priority, effectivePriority, waitingResource which corresponding to the thread it represents, the priority of that thread, the effective priority of that thread, and the resource this thread is waiting for.
- (modification)A member variable resources implemented by a binary search tree, which holds all the resources acquired by this thread.

5.4 Description

5.4.1 Scheduler

```
procedure getPriority(thread)
    return thread.schedulingState.priority
end procedure
procedure getEffectivePriority(thread)
    return thread.schedulingState.effectivePriority
end procedure
procedure setPriority(thread, p)
    if p < priorityMinimum OR p > priorityMaximum then
        return
    end if
    thread.schedulingState.setPriority(p)
end procedure
```

5.4.2 LotteryThreadQueue

```
procedure INITIALIZE()
    resourceHolder ← null;
    waiters ← new empty TreeSet.
end procedure
procedure UPDATE(TMP)
    if tmp != sumPriority then
        if resourceHolder != null then
            resourceHolder.updateResource(this,maxPriority)
        else
            sumPriority ← tmp
        end if
    end if
end procedure
```

```
procedure UPDATEWAITER(STATE, EP)
    tmp ← sumPriority
    waiters.remove(state)
    tmp ← tmp - state.effectivePriority
    state.effectivePriority ← EP
    watiers.add(state)
    tmp ← tmp + state.effectivePriority
    update(tmp)
end procedure
procedure WAITFORACCESS(THREAD)
    state ← thread.schedulingState
    tmp ← sumPriority
    state.waitingResource ← this
    waiterQueue.add(state)
    tmp ← tmp + state.effectivePriority
    update()
end procedure
```

```
procedure ACQUIRE(THREAD)
    state ← thread.schedulingState
    resourceHolder ← state
    state.addResource(this)
end procedure
```

```
procedure pickNextThread()
    rnd ← random number form 0 to sumPriority-1
    for i in waiters do
        rnd ← rnd - i.effectivePriority
        if rnd < 0 then
            return i.thread
        end if
    end for
end procedure
```

```
procedure nextThread()
    if resourceHolder != null then
        resourceHolder.removeResource(this)
        resourceHolder ← null
    end if
    state ← pickNextThread()
    if state == null then
        return null
    end if
    thread ← state.thread
    update()
    state.waitingResource = null
    state.addResource(this); return thread
end procedure
```

5.4.3 SchedulingState

```
procedure Initialize()
    priority, effectivePriority ← priorityDefault
    resources ← empty TreeSet
    waitingResource ← null
end procedure
```

```
procedure update(tmp)
    if tmp != effectivePriority then
        if waitingResource != null then
            waitingResource.updateWaiter(this,tmp)
        else
            effectivePriority ← tmp
        end if
    end if
end procedure
```

```
procedure setPriority(p)
    tmp ← effectivePriority
    tmp ← tmp - priority
    priority ← p
    tmp ← tmp + priority
    update(tmp)
end procedure
procedure updateResource(res, maxP)
    resources.remove(res)
    tmp ← effectivePriority
    tmp ← tmp - res.sumPriority
    res.maxPriority ← maxP
    resources.add(res)
    tmp ← tmp + res.sumPriority
    update(tmp)
end procedure
procedure addResource(res)
    tmp ← effectivePriority
    resources.add(res)
    tmp ← tmp + res.sumPriority
    update(tmp)
end procedure
procedure removeResource(res)
    tmp ← effectivePriority
    resources.remove(res)
    tmp ← tmp - res.sumPriority
    update(tmp)
end procedure
```

5.5 Description of Tests

Random Test

Set several threads such that the i-th has priority 10^i.

Then there is a very big chance that they are performed in the decreasing order of priority. Run it multiple times to see if it the cases.

Priority Donation Test

Check whether priority donation works by set up a condition, and see when a process release that condition, whether or nor its priority is recovered.