Nachos Phase 2

1. 实现文件系统调用，内部实现已经都有了，主要是handle syscall然后转发请求就是了。唯一需要实现的就是file descriptor机制。这里使用最简单的方法：定长数组记录file descriptor到内部的OpenFile的对应，而查找新的file descriptor也不过是array iteration而已：

// 返回一个空闲的fd, 如果没有返回-1

protected int getFileDescriptor()

{

for (int i = 0; i < MAX\_OPEN\_FILE; ++i)

{

if (files[i] == null) return i;

}

return -1;

}

private OpenFile[] files = new OpenFile[MAX\_OPEN\_FILE];

然后随便帖一个典型的函数实现：

protected int doRead(int fd, int bufferAddr, int count)

{

if (files[fd] == null) return -1;

if (count < 0) return -1;

byte[] bytes = new byte[count];

int bytesRead = files[fd].read(bytes, 0, count);

if (bytesRead == -1) return -1;

int bytesWritten = writeVirtualMemory(bufferAddr, bytes, 0, bytesRead);

if (bytesWritten != bytesRead) return -1;

return bytesRead;

}

再来一个OpenCreate(其中要分配fd):

protected int doOpenCreate(int nameAddr, boolean create)

{

int fd = getFileDescriptor();

if (fd == -1) return -1;

String filename = readVirtualMemoryString(nameAddr, MAX\_STRING\_LENGTH);

if (filename == null) return -1;

files[fd] = UserKernel.fileSystem.open(filename, create);

if (files[fd] == null) return -1;

return fd;

}

测试: 最后写了个C程序来测试调用。(其中还有multi-programming的测试内容，故main函数等后面再写。其中的test id对应grading2.html里面的编号)

#include "syscall.h"

#include "stdio.h"

#include "stdlib.h"

#define CPYBUFSZ 100

int fs\_test() {

// testID 1 : calls syscall creat/close/unlink and checks that they work

const char filename[] = "test.file";

int fd = creat(filename);

assert(fd == 2);

assert(sizeof(filename) == write(fd, filename, sizeof(filename)));

printf("type a char: ");

char c = getch();

assert(1 == write(fd, &c, 1));

close(fd);

fd = open(filename);

char buffer[sizeof(filename)];

c = 0;

assert(fd == 2);

assert(sizeof(filename) == read(fd, buffer, sizeof(filename)));

printf("%s\n", buffer);

assert(1 == read(fd, &c, 1));

printf("%c\n", c);

close(fd);

unlink(filename);

// testID 3 : tests if your syscall open fails gracefully when stubFileSystem's openfile limit's exceeded

const char filename2[] = "test.file ";

int i;

for (i = 2; i < 16; ++i) {

sprintf(filename2, "test.file%d", i);

assert(creat(filename2) >= 0);

}

assert(creat(filename) == -1);

for (i = 2; i < 16; ++i) {

assert(close(i) != -1);

}

for (i = 2; i < 16; ++i) {

sprintf(filename2, "test.file%d", i);

assert(unlink(filename2) != -1);

}

// testID 7 : copies between files, tests creat, open, read, write, close

int src = open("proj2-test.c");

int dst = creat("proj2-test.dup");

char cpybuf[CPYBUFSZ];

while ((i = read(src, cpybuf, CPYBUFSZ)) > 0) {

assert(write(dst, cpybuf, i) == i);

}

close(src);

close(dst);

// testID 8 : tests that write fails gracefully on bad arguements (e.g. bad address)

// testID 9 : tests that read fails gracefully on bad arguments (e.g. writing back to a readonly part of virtual memory)

fd = open("proj2-test.dup");

assert(-1 == write(fd, (void\*) 0x7FFE1234, 0x7FFFFFFF));

assert(-1 == read(fd, (void \*) 0x00000000, 100));

close(fd);

}

测试结果: 程序很简单，看一下就知道什么意思，现在已经忘记具体细节了，总之结果当时是正确的。另外也可以用现有tests目录下的coff文件来测试。

2. 实现多道程序。主要是实现read virtual memory和write virtual memory, 以及在load sections的时候做一些调整。原始的代码都是物理地址，所以不支持多道程序。这里让每个process都用自己的pageTable来做映射。CPU那边，由于这道题的关系直接支持了pageTable也就不必关心了:

首先是read/write virtual memory. 假设那张表存在的情况下很容易写：

private TranslationEntry[] pageTable;

public int writeVirtualMemory(int vaddr, byte[] data, int offset,

int length)

{

Lib.assert(offset >= 0 && length >= 0 && offset + length <= data.length);

byte[] memory = Machine.processor().getMemory();

int vpn = vaddr / pageSize;

int pageOffset = vaddr % pageSize;

int bytesLeft = length;

while (bytesLeft > 0 && 0 <= vpn && vpn < pageTable.length && !pageTable[vpn].readOnly)

{

int amount = Math.min(bytesLeft, pageSize - pageOffset);

pageTable[vpn].used = true;

pageTable[vpn].dirty = true;

int ppn = pageTable[vpn].ppn;

int paddr = ppn \* pageSize + pageOffset;

System.arraycopy(data, offset, memory, paddr, amount);

Lib.assert(pageTable[vpn].vpn == vpn);

++vpn; pageOffset = 0;

bytesLeft -= amount; offset += amount;

}

return length - bytesLeft;

}

其次就是那张表的维护了，首先维护一张free pages的链表，然后写loadSections:

package nachos.userprog;

import java.util.LinkedList;

import nachos.machine.TranslationEntry;

import nachos.threads.Lock;

public class MemoryManager

{

public MemoryManager(int n)

{

lock.acquire();

freePages = new LinkedList();

for (int i = 0; i < n; ++i)

{

freePages.addLast(new Integer(i));

}

lock.release();

}

/\*\*

\* Allocate n pages and return their physical page number.

\* @param vpn the start virtual page number

\* @param n the number of pages to be allocated

\* @return the translation entry of allocated pages, null if failed

\*/

public TranslationEntry[] allocPages(int vpn, int n)

{

lock.acquire();

if (freePages.size() < n)

{

lock.release();

return null;

}

TranslationEntry[] pages = new TranslationEntry[n];

for (int i = 0; i < n; ++i)

{

int ppn = ((Integer)freePages.removeFirst()).intValue();

pages[i] = new TranslationEntry(vpn + i, ppn, false, false, false, false);

}

lock.release();

return pages;

}

public void deallocPages(TranslationEntry[] pages)

{

if (pages == null) return;

lock.acquire();

for (int i = 0; i < pages.length; ++i)

{

if (pages[i].valid)

freePages.addLast(new Integer(pages[i].ppn));

}

lock.release();

}

public int freepageCount()

{

return freePages.size();

}

private Lock lock = new Lock();

private LinkedList freePages;

}

protected boolean loadSections()

{

pageTable = UserKernel.memoryManager.allocPages(numPages);

if (pageTable == null)

{

coff.close();

Lib.debug(dbgProcess, "\tinsufficient physical memory");

return false;

}

int vpn = 0;

// load sections

for (int s = 0; s < coff.getNumSections(); s++)

{

CoffSection section = coff.getSection(s);

Lib.debug(dbgProcess, "\tinitializing " + section.getName()

+ " section (" + section.getLength() + " pages)");

boolean readOnly = section.isReadOnly();

for (int i = 0; i < section.getLength(); i++)

{

vpn = section.getFirstVPN() + i;

section.loadPage(i, pageTable[vpn].ppn);

pageTable[vpn].readOnly = readOnly;

pageTable[vpn].valid = true;

}

}

for (vpn++; vpn < numPages; ++vpn)

{

pageTable[vpn].valid = true;

}

return true;

}

然后在unloadSections里面调用dealloc就是了。

测试：可以用原有的程序，还写了一段Java的selfTest来测试跨page的read/write virtual memory.

public static void selfTest()

{

// testID 12 : tests readVirtualMemory with a large valid range, make sure read right data

// testID 13 : tests writeVirtualMemory with a large valid range, make sure wrote right data

UserProcess proc = new UserProcess();

proc.pageTable = UserKernel.memoryManager.allocPages(2);

proc.pageTable[0].valid = true;

proc.pageTable[1].valid = true;

byte[] bytes = new byte[2\*pageSize];

for (int i = 0; i < pageSize / 2; ++i)

{

Lib.bytesFromInt(bytes, 4 \* i, i);

}

Lib.assert(proc.writeVirtualMemory(0, bytes, 2 \* pageSize - 400, 400) == 400);

Lib.assert(proc.writeVirtualMemory(400, bytes, 0, 2 \* pageSize - 400) == 2 \* pageSize - 400);

bytes = new byte[2\*pageSize];

Lib.assert(proc.readVirtualMemory(0, bytes, 2 \* pageSize - 400, 400) == 400);

Lib.assert(proc.readVirtualMemory(400, bytes, 0, 2 \* pageSize - 400) == 2 \* pageSize - 400);

for (int i = 0; i < pageSize / 2; ++i)

{

Lib.assert(i == Lib.bytesToInt(bytes, 4 \* i));

}

++processCount;

proc.doExit(false, 0);

--processCount;

}

结果正常。

3. 实现exec等系统调用。有了上面的mp支持，这个也不过是转发而已。就是Process类要添加几个成员变量来记录有些Process状态。

private static UserProcess root = null;

private UserProcess parent = null;

private Map children = new TreeMap();

private int exitStatus = 0;

private boolean abnormalExit = false;

private int processID;

private UThread thread;

private static int nextProcessID = 1;

private static int processCount = 0;

变量意义比较显然，不赘述了。随便帖一个函数：

protected int doExec(int fileAddr, int argc, int argvAddr)

{

if (argc < 0) return -1;

String filename = readVirtualMemoryString(fileAddr, MAX\_STRING\_LENGTH);

if (filename == null) return -1;

String[] args = new String[argc];

for (int i = 0; i < argc; ++i, argvAddr += 4)

{

Integer argiAddrI = readVirtualMemoryInteger(argvAddr);

if (argiAddrI == null) return -1;

int argiAddr = argiAddrI.intValue();

args[i] = readVirtualMemoryString(argiAddr, MAX\_STRING\_LENGTH);

if (args[i] == null) return -1;

}

UserProcess proc = newUserProcess();

if (!proc.execute(filename, args))

{

proc.doExit(true, 0);

return -1;

}

addChild(proc);

return proc.processID;

}

测试：主要靠tests下的那些文件来进行，其次还有一个C函数：

void mp\_test() {

// testID 3 : tests exec with error arguments (e.g. bad file name)

assert(-1 == exec("hahaha.haha", 0, 0));

// test that halt don't work for non-root thread

halt();

// testID 4 : tests your syscall join to a non-child

int retval;

assert(-1 == join(1, &retval)); // 1 should be the pid of shell or myself

}

结果正常。

4. 彩票算法调度器

帖一些核心的东西：

public KThread nextThread() {

Lib.assert(Machine.interrupt().disabled());

if (groups[0] == null) return null;

int winner = Lib.random(getTotalTickets());

LotteryThreadState candidate = (LotteryThreadState)groups[0];

do {

winner -= candidate.totalTickets;

if (winner < 0) {

candidate.unlink();

acquire(candidate.thread);

return candidate.thread;

}

candidate = (LotteryThreadState)candidate.next;

} while (candidate != groups[0]);

Lib.assertNotReached();

return null;

}

protected void adjustTickets(int num) {

if (num == 0) return;

LotteryQueue queue = this;

do {

queue.maxGroup += num;

if (queue.holder != null) {

((LotteryThreadState)queue.holder).totalTickets += num;

queue = (LotteryQueue)queue.holder.queue;

} else {

queue = null;

}

} while (queue != null);

}

其他和以前的也差不多。测试也沿用以前的测试，有一定随机性，但总体符合预期并且没有异常等就行。