

四川 大 学

操作系统实验课程

--Nachos 部分实验报告

学 院：_____

专 业：_____

年 级：_____

组 员：_____

指导教师评阅意见：_____

指导教师评阅成绩：_____

提交时间 2023 年 11 月 7 日

1. 实验项目一：Nachos 系统调用实验

1.1 实验目的：

完成 Nachos 的异常处理流程，掌握操作系统中对异常，特别是系统调用流程的实现。

1.2 实验环境：

描述所用实验环境,包括计算机硬件和软件资源的情况,如选用的操作系统、机器配置、编译器版本等。

选用的操作系统: VMware 16.0 -Ubuntu Linux 22.04.2 -NachOS-4.1

机器配置:

虚拟机设置

设备	摘要
内存	4 GB
处理器	8
硬盘 (SCSI)	30 GB
CD/DVD (SATA)	正在使用文件 ubuntu-22.04....
网络适配器	自定义 (VMnet8 (NAT))
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

编译器版本: GCC -4.85, G++ -4.85

1.3 实验内容：

对实践过程的详细说明，如对 Nachos 平台的哪些代码进行了什么样的修改，采用了何种算法或思想等。

List 核心的代码，如以文件为单位一一进行描述。

可以结合适当的流程图或者类图来辅助描述。

Exception.cc

Void increment():

函数目的：实现 PC 寄存器的递增

1. 将当前 PC 寄存器 (PCReg) 的值存入 PrevPC 寄存器 (PrevPCReg)，保存上一次执行的指令
2. 通过将 NextPC 寄存器 (NextPCReg) 中的值存入 PC 寄存器 (PCReg)，使 PC 寄存器指向下一条将要执行的指令
3. 通过将 NextPC 寄存器 (NextPCReg) ++，使 NextPC 寄存器指向 PC 寄存器执行

完当前存储的指令后的下一跳指令

```
51 void
52 incrementPC() {
53     /* set previous programm counter (debugging only)*/
54     kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
55
56     /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
57     kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(NextPCReg));
58
59     /* set next programm counter for brach execution */
60     kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(NextPCReg) + 4);
61 }
```

ExceptionHandler => case SC_Write:

1. 从 4、5、6 号寄存器中，读入输入的三个参数：
 - a) 需要写入的字符串的地址（指向字符串第一位字符的地址）
 - b) 需要写入的字符串大小
 - c) 需要写入的位置（ConsoleOutput）
2. 通过调用 SysWrite 函数来实现系统调用写入
3. 将 SysWrite 返回的结果写入到 2 号寄存器，作为返回值

```
110 case SC_Write:
111     /* Debug notation. */
112     DEBUG(dbgSys, "Write from buffer to consoleOutput" << kernel->machine->ReadRegister(4)
113           << kernel->machine->ReadRegister(5)
114           << kernel->machine->ReadRegister(6) << "\n");
115
116     /* Read from registers. */
117     int addr;
118     addr = (int) kernel->machine->ReadRegister(4);
119     int size;
120     size = (int) kernel->machine->ReadRegister(5);
121     OpenFileId output;
122     output = (OpenFileId) kernel->machine->ReadRegister(6);
123
124     /* System write and load the writing result onto register 2. */
125     kernel->machine->WriteRegister(2, SysWrite(addr, size, output));
126
127     /* Increment PC register by 1, and return the system writing result. */
128     incrementPC();
129     return;
130     ASSERTNOTREACHED();
131     break;
```

ExceptionHandler => case SC_Read:

1. 从 4、5、6 号寄存器中，读入输入的三个参数：
 - a) 需要读入到的位置
 - b) 需要读入的字符串大小
 - c) 从哪里读入？ => (ConsoleInput)
2. 通过调用 SysRead 函数来实现系统调用写入
3. 将 SysRead 返回的结果写入到 2 号寄存器，作为返回值

```

133     case SC_Read:
134         /* Debug notation. */
135         DEBUG(dbgSys, "Read file to buffer" << kernel->machine->ReadRegister(4) << ", "
136                << kernel->machine->ReadRegister(5) << "words" << "\n");
137
138         /* Read from registers. */
139         addr = (int) kernel->machine->ReadRegister(4);
140         size = (int) kernel->machine->ReadRegister(5);
141         OpenFileId input;
142         input = (OpenFileId) kernel->machine->ReadRegister(6);
143
144         /* Call System read and load the result onto register 2. */
145         result = SysRead(addr, size, input);
146         kernel->machine->WriteRegister(2, (int) result);
147
148         /* Increment PC register by 1, and return the system writing result. */
149         incrementPC();
150         return;
151         ASSERTNOTREACHED();
152         break;

```

ExceptionHandler => case SC_Exec:

1. 从 4 号寄存器中，读入输入的参数：
 - a) 需要读入的字符串的地址（指向字符串第一位字符的地址）
2. 通过调用 SysExec 函数来实现系统调用执行
3. 将 SysExec 返回的结果（分支出的子进程名）写入到 2 号寄存器，作为返回值

```

154     case SC_Exec:
155         /* Debug notation. */
156         DEBUG(dbgSys, "execute a command" << kernel->machine->ReadRegister(4) << "\n");
157
158         /* Read from register 4 => the address of the input buffer (input command). */
159         addr = kernel->machine->ReadRegister(4);
160
161         /* Call System Execute and load the child process id onto register 2. */
162         pid_t child;
163         child = SysExec(addr);
164         kernel->machine->WriteRegister(2, (int) child);
165
166         /* Increment PC register by 1, and return the system writing result. */
167         incrementPC();
168         return;
169         ASSERTNOTREACHED();
170         break;

```

ExceptionHandler => case SC_Exec:

1. 从 4 号寄存器中，读入输入的参数：
 - a) 子进程的进程名
2. 通过调用 SysJoin 函数来实现系统调用进程同步
3. 将 SysJoin 返回的结果写入到 2 号寄存器，作为返回值

```

172     case SC_Join:
173         /* Debug notation. */
174         DEBUG(dbgSys, "join" << kernel->machine->ReadRegister(4) << "\n");
175
176         /* Read from register 4 => the input child process id. */
177         child = (pid_t) kernel->machine->ReadRegister(4);
178
179         /* Call System Join to wait the process to be executed. */
180         result = SysJoin(child);
181         kernel->machine->WriteRegister(2, (int) result);
182
183         /* Increment PC register by 1, and return the system writing result. */
184         incrementPC();
185         return;
186         ASSERTNOTREACHED();
187         break;

```

Ksyscall.h

Int SysWrite(int addr, int size, OpenFileId output):

1. 通过输入的 addr（需要写入的字符串地址，第一个字符的地址）和 size（需要写入的字符串长度），将字符串从给定地址中读取到 buffer 中
2. 通过 write 函数，将 buffer 中的字符串输出
3. 将调用 write 得到的返回值返回

```
39 int SysWrite(int addr, int size, OpenFileId output) {
40     /* Load memory into buffer. */
41     char buffer[BUF_SIZE];
42     for (int i = 0; i < size; ++i) {
43         kernel->machine->ReadMem(addr, 1, (int *) &buffer[i]);
44         addr++;
45     }
46     /* Return result of write. */
47     return write(output, buffer, (size_t) size);
48 }
```

Int SysRead(int addr, int size, OpenFileId input):

1. 先通过调用 read 函数，将输入的值字符串读入到 buffer 中
2. 通过输入的 addr（需要写入的字符串地址，第一个字符的地址）和 size（需要写入的字符串长度），将字符串 buffer 中写入到指定位置中

```
50 int SysRead(int addr, int size, OpenFileId input) {
51     /* Load input into buffer (as an array). */
52     char buffer[BUF_SIZE];
53     read(input, buffer, (size_t) size);
54     /* Write buffer into the given address. */
55     if (input == ConsoleInput) {
56         int i = 0;
57         while (i < size && buffer[i] != '\0') {
58             kernel->machine->WriteMem(addr, 1, buffer[i]);
59             addr++, i++;
60         }
61         kernel->machine->WriteMem(addr, 1, (char) NULL);
62     }
63     return 1;
64 }
```

Int SysExec(int addr):

1. 通过输入的 addr，将需要执行的指令加载到 buffer 中
2. 通过 vfork, execl 命令为加载的指令分支新的进程，并将子进程 id 返回

```

67 SpaceId SysExec(int addr) {
68     /* Load command (char array form) into buffer. */
69     char buffer[BUF_SIZE];
70     int ch;
71     int i = 0;
72
73     do {
74         kernel->machine->ReadMem(addr, 1, (int *) &ch);
75         buffer[i] = *((char *) &ch);
76         addr++, i++;
77     } while (*((char *) &ch) != '\0');
78
79     /* Fork the process to execute. */
80     pid_t child;
81     child = vfork();
82
83     if (child == 0) {
84         execl("/bin/sh", "/bin/sh", "-c", buffer, NULL);
85         _exit(EXIT_FAILURE);
86     } else if (child < 0) {
87         return EPERM;
88     }
89     return (SpaceId) child;
90 }

```

Int SysJoin(SpaceId id):

1. 通过输入的进程 id, 调用 waitpid 函数等待进程执行完毕

```

93 int SysJoin(SpaceId id) {
94     return waitpid((pid_t) id, (int *) 0, 0);
95 }

```

1.4 实验结果:

截图说明运行效果, 并且辅助相关描述信息。

1. `ls` 和 `ls -l` 命令执行测试

```
filtee@iZ2vczcz19gseqovr8jso8Z: ~/Desktop/NachOS/code/test
filtee@iZ2vczcz19gseqovr8jso8Z:~/Desktop/NachOS/code/test$ ../build.linux/nachos -x shell.noff

tests summary: ok:0
--ls
add.c      halt.c      Makefile.dep  script      shell.noff   sort.noff   write.c
add.noff   halt.noff    matmult.c     segments.c  SOCKET_0     start.o     I
DISK_0     Makefile     matmult.noff  shell.c     sort.c       start.s
--ls -l
total 88
-rw-rw-r-- 1 filtee filtee 266 Nov 4 13:26 add.c
-rw-rw-r-- 1 filtee filtee 500 Nov 7 14:27 add.noff
-rw-rw-r-- 1 filtee filtee 131076 Nov 4 13:26 DISK_0
-rw-rw-r-- 1 filtee filtee 541 Nov 4 13:26 halt.c
-rw-rw-r-- 1 filtee filtee 484 Nov 7 14:27 halt.noff
-rw-rw-r-- 1 filtee filtee 1337 Nov 4 13:26 Makefile
-rw-rw-r-- 1 filtee filtee 2058 Nov 4 13:26 Makefile.dep
-rw-rw-r-- 1 filtee filtee 824 Nov 4 13:26 matmult.c
-rw-rw-r-- 1 filtee filtee 740 Nov 7 14:27 matmult.noff
-rw-rw-r-- 1 filtee filtee 507 Nov 4 13:26 script
-rw-rw-r-- 1 filtee filtee 1394 Nov 4 13:26 segments.c
-rw-rw-r-- 1 filtee filtee 517 Nov 4 13:26 shell.c
-rw-rw-r-- 1 filtee filtee 580 Nov 7 14:27 shell.noff
srwxrwxr-x 1 filtee filtee 0 Nov 7 14:53 SOCKET_0
-rw-rw-r-- 1 filtee filtee 1315 Nov 4 13:26 sort.c
-rw-rw-r-- 1 filtee filtee 676 Nov 7 14:27 sort.noff
-rw-rw-r-- 1 filtee filtee 4012 Nov 7 14:27 start.o
-rw-rw-r-- 1 filtee filtee 3591 Nov 4 13:26 start.s
-rw-rw-r-- 1 filtee filtee 376 Nov 4 13:26 write.c
```

2. mkdir、rm 和 make 等命令执行

```
filtee@iZ2vczcz19gseqovr8jso8Z: ~/Desktop/NachOS/code/test
filtee@iZ2vczcz19gseqovr8jso8Z:~/Desktop/NachOS/code/test$ ../build.linux/nachos -x shell.noff

tests summary: ok:0
--mkdir some
--ls
add.c      DISK_0     halt.noff  Makefile.dep  matmult.noff  segments.c  shell.noff  some  sort.noff  start.s
add.noff   halt.c     Makefile   matmult.c     script        shell.c     SOCKET_0    sort.c  start.o   write.c
--rm -rf some
--ls
add.c      halt.c      Makefile.dep  script      shell.noff   sort.noff   write.c
add.noff   halt.noff    matmult.c     segments.c  SOCKET_0     start.o
DISK_0     Makefile     matmult.noff  shell.c     sort.c       start.s
--make clean
rm -f *.o *.ii
rm -f *.coff *.noff
--make
/usr/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/cpp -I../userprog -I../lib start.s > strt.s
/usr/local/nachos/bin/decstation-ultrix-as -mips2 -o start.o strt.s
rm -f strt.s
/usr/local/nachos/bin/decstation-ultrix-gcc -G 0 -O3 -ggdb -c -I../userprog -I../lib -c add.c -o add.o
/usr/local/nachos/bin/decstation-ultrix-ld -T script -N start.o add.o -o add.coff
/usr/local/nachos/bin/decstation-ultrix-strip add.coff
../coff2noff/coff2noff add.coff add.noff
numsections 3
Loading 3 sections:
".text", filepos 0xd0, mempos 0x0, size 0x1c0
".data", filepos 0x290, mempos 0x200, size 0x0
".bss", filepos 0x0, mempos 0x200, size 0x0
```

1.5 实验总结:

总结本实验的完成情况，包括代码是否编写完成，是否调试通过，能否正常运行，实现了实验要求中要求的哪些项（对实验要求的满足程度）；

总结本实验所涉及到的知识点。

实验虽然有许多波折，但是 XXX 同学在进行了众多尝试和实验后成功完成了对 exception.cc 和 ksyscall.h 等代码的修改，调试通过，完成了对 shell.noff 的正常运行。

实现了实验要求中对 Nachos 的异常处理，掌握了对系统调用流程的实现。并列出了对 Nachos 平台的已修改代码进和采用的算法或思想。