OS A1

by Nidhi Surya Prakash

Submission date: 15-Sep-2018 12:00AM (UTC-0400)
Submission ID: 1002258940
File name: CIS_657_Lab_1_answers_Nidhi_OS_A1.pdf (535.95K)
Word count: 2471
Character count: 12390

CIS 657 Lab 1

Principles of Operating Systems

NIDHI SURYA PRAKASH

SUID:215895619

Download Nachos.tar from the blackboard and unpack it. Answer all of the following questions by examining the Nachos code (all sub-directories of code directory).

(1) **(5pts)** What are the possible NachosProcess (called Thread) states and where are they defined (filename & line number)?

Answer:

```
enum ThreadStatus {JUST_CREATED, RUNNING, READY, BLOCKED};
```

NachosProcess state	Description	Filename	Line number
READY	In this state, the thread is on a queue like list called ready list where it is eligible to receive control of the resource (CPU) even though at that moment another thread is still running; only threads found in the READY state would be found on the ready list. The scheduler responsible for scheduling of these threads, selects a thread for execution from the list and changes the state of thread from READY to RUNNING.	thread.h	63
RUNNING	Once the thread changes its state from READY to RUNNING, the global variable currentThread points to this currently running thread. Only 1 thread can be in this state at a time.	thread.h	63
	As the name says, the thread gets blocked from execution until a particular external event	thread.h	63

BLOCKED	completes, the blocked thread cannot be on the ready list. Here, the thread puts itself to sleep using the Thread::Sleep() function or operation; while waiting on some condition variable, semaphore.		
JUST_CREATED	In this state, the thread is said to exist but has no stack allotted to it yet, it's a temporary state which is mostly used for thread creation. Thread constructor is responsible for creating a thread but Thread::Fork() makes this thread into a thread that the CPU accepts and can execute.	thread.h	63

(2) **(5 pts)** In threadtest.cc, how many lines are printed when you change:

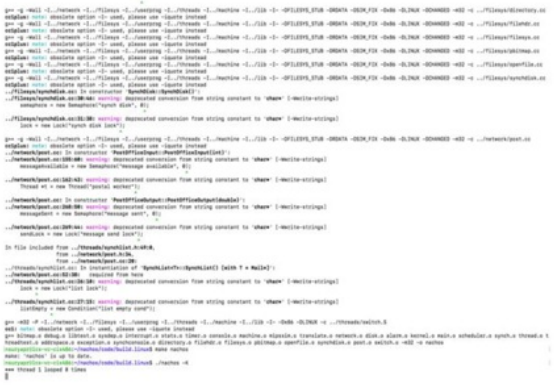
```
for(int num = 0; num < 5; num++)
```

to

```
for(int num = 0; num < which; num++)
```

Why is that?

Output:



- One time

"Thread 1 looped 0 times"

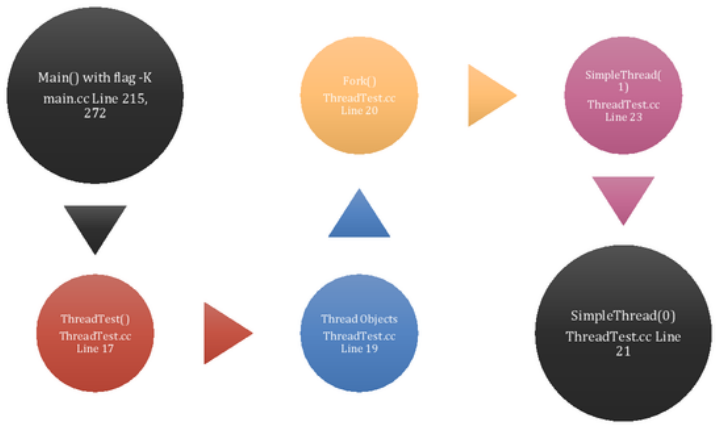
Reason: When the function threadtest.cc is executed, the control first goes to ThreadTest() function. Here, in the first line a thread 't' is created which then is forked in the second line of the function. The fork function has SimpleThread as one of its arguments. The SimpleThread function is defined in the same file -> threadtest.cc and accepts one argument. In the fork argument, SimpleThread is called with an argument equal to 1.

When SimpleThread is called with 1 as the argument, which takes the value 1 and num has the value 0. So the print statement outputs: "*** thread 1 looped 0 times".As the for loop continues, num is incremented and since to enter the loop num should be less than 'which', it does not run the code.

Now, the control goes back to ThreadTest(), where SimpleThread is called again with 'which' equal to 0. As we know num should be less than the 'which' value, it does not enter the for loop to run the code.

(3) **(10 pts)** Draw the call graph of creating a new NachoProcess with SimpleThread function.

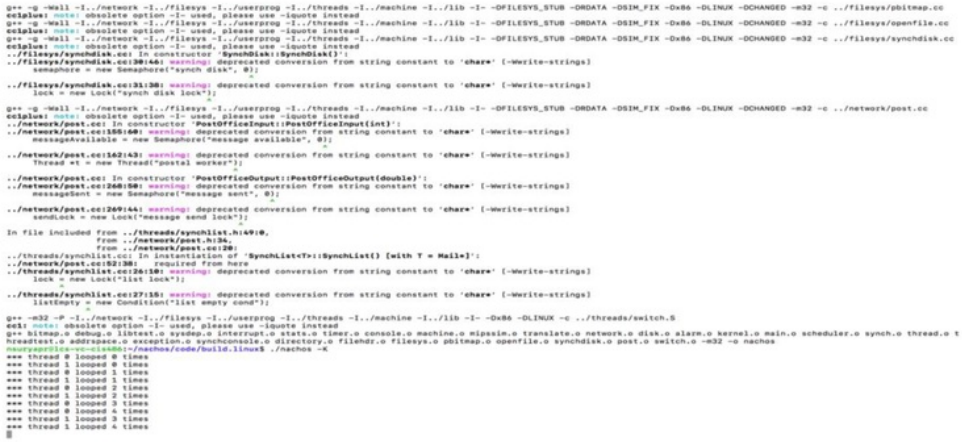
Answer:



Execution of the program begins with control at main.cc file. As we enter int main() function, the flag threadTestFlag is initialized to False. After all the initializations are done, we encounter a 'for' loop, the else-if condition on line 215 satisfies the conditions and hence sets the threadTestFlag to True. Once we're out of the for-loop, we go through kernel initialization and then an if condition that checks for threadTestFlag value. Since it is set, the ThreadTest() function is called.

Now we begin with control at threadtest.cc which contains the function ThreadTest() at line 17. First, thread creation occurs using thread objects at line 19 followed by forking that occurs at line 20. Now we encounter the function call to SimpleThread() with 1 as the argument. After the function is executed, we again encounter another function call to SimpleThread on line 21 but with 0 as the argument.

The output is as follows:



(4) **(25 pts)** Using flags will help you design and test your operating system (later in other labs and programming assignments).

To use a flag: **./nachos -flag <flag parameters>**

Examine source code for all the predefined flags and discuss briefly how Nachos operates and what information is displayed if it exists for each flag and its parameter(s).

Answer:

Flag Name	Description	Usage	Operation/Implementation in program
-d	Debugs messages that required to be printed.	nachos -d <debugflags>	In file debug.h, all the pre-defined debugging flags are available from line 22. In main.cc , this flag is implemented on line 202. The debugging routines basically allow the users to turn on selected debugging messages, controllable from the command line arguments passed to Nachos (-d). Used along with/ parameters used with this flag are:

			'+' → debug messages are turned on 't' → thread system 's' → semaphores, locks & conditions 'i' → interrupt emulation 'm' → machine emulation (USER_PROGRAM) 'd' → disk emulation (FILESYS) 'f' → file system (FILESYS) 'a' → address spaces (USER_PROGRAM) 'n' → network emulation (NETWORK) Examples from our code – dbgAll, dbgSys, dbgNet, dbgDisk etc.,
-rs	This flag causes Yield to occur at random and repeatable spots	nachos -rs <random seed #>	
-z	Prints copyright messages	nachos -z	Used/set in main.cc on line 207 A string comparison is taking place after which, ⇨ cout << copyright << "\n"; basically, printing occurs.
-s	This flag is responsible in causing execution of user program in single-step mode	nachos -s	
-x	Runs user programs (running or loading a file). Eg: - ./nachos -x ~/test/halt	nachos -x <nachos file>	Used/set in line 210 in main.cc A string comparison takes place after which the ASSERT function is called.
-ci	Specifying the file for console input. Default: stdin	nachos -ci <consoleIn>	
-co	Specifying the file for console output. Default: stdout	nachos -co <consoleOut>	
-n	It sets the network reliability	nachos -n <network reliability>	Used/set in the files under folder network to check if it is enabled in post.cc
-m	This flag sets this machine's host id (needed for the network)	nachos -m <machine id>	
-K	self-test of kernel threads and synchronization is made to run. Eg: ./nachos -K	nachos -K	In our program, ths flag sets threadTestFlag to TRUE and also invokes the function ThreadTest(). It then creates new threads by calling the SimpleThread function in ThreadTest.cc.It leads to the output that provides us with information of threads created and how many times the code went through a loop. Used/shown on line 215 in main.cc.
-C	Interactive console test is made to run	nachos -C	This flag sets the consoleTestflag to TRUE which invokes the kernel

			function ConsoleTest() (defined at Line 169 in Kernel.cc). Leads to testing of console I/0 by continuously echoing anything that is typed until an interrupt ^D is passed.
-N	Runs a two-machine network test	nachos -N	This flag sets the networkTestFlag to TRUE which then invokes the kernel function NetworkTest() (defined at Line 200 of Kernel.cc). A messaging system is built/stimulated between 2 network computers and each one sends a mail that contains "To" and "From" after which it sends the message and waits for the acknowledgement.
-f	Physical disk is formatted using this flag	nachos -f	
-cp	This flag copies files from UNIX OS to NachOS	nachos -cp <unix file><nachos file>	Set/used in line 225 in main.cc It opens the UNIX file, checks length of file after which it creates nachos file of same length, copies the data in chunks and finally the closes the file.
-p	Used to print Nachos file to stdout(Standard output) Parameter: nachos file name => nachos -p <nachos-file>	nachos -p <nachos file>	Set/used in line 231 in main.cc.
-r	This flag on usage will be responsible for removing Nachos file from the file system	nachos -r <nachos file>	Set/shown on line 236 in main.cc. It calls the Remove function that is defined in filesys.cc.
-l	Listing of contents from Nachos Directory is possible using this flag	nachos -l	Set/shown on line 241 in main.cc. It sets the dirListFlag to true after which it then lists all the files that are present in that directory. It is defined in filesys.cc where it prints all the current directory entries.
-D	This flag prints contents of entire file system	nachos -D	Set/shown on line244 in main.cc. This flag sets the dumpFlag to true and then invokes the kernel function Print() which prints everything in the Nachos filesystem.

(5) **(10pts)** What system calls are currently implemented in Nachos? Explain what it does in detail and find a corresponding UNIX system call if exists.

Answer:

The system calls currently implemented in Nachos are Halt() and Add(). But the Nachos program code currently defines 17 system calls:

- ⇨ Halt
- ⇨ Exit
- ⇨ Exec
- ⇨ Join
- ⇨ Create
- ⇨ Remove
- ⇨ Open
- ⇨ Read
- ⇨ Write
- ⇨ Seek
- ⇨ Close
- ⇨ ThreadFork
- ⇨ ThreadYield
- ⇨ ExecV
- ⇨ ThreadExit
- ⇨ ThreadJoin
- ⇨ Add

The Halt() system call as SysHalt() and Add() as SysAdd() functions. Both are defined in the ksyscall.h file. The halt stops the machine running NachOS and further provides the performance stats whereas, the Add system call adds the two operands provided to it as arguments. The NachOS halt command has an equivalent command used in UNIX which is halt. The system call defined is ksyscalls.h have the following code:

```
void SysHalt()
{
 kernel->interrupt->Halt();
}
int SysAdd(int op1, int op2)
{
```

```
    return op1 + op2;
}
```

(6) **(15pts)** What would happen when you execute the halt program (code/test/halt.c) - ./nachos -x ~/test/halt? Explain why it happens. Draw the call graph from a user program (halt) instruction to running a system call implementation.

Answer:

Below is the screenshot of the execution of the command: ./nachos -x ~/test/halt.

```
                from ../network/post.cc:20:
../threads/synchlist.cc: In instantiation of 'SynchList<T>::SynchList() [with T = Mail*]':
../network/post.cc:52:38:   required from here
../threads/synchlist.cc:26:10: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
     lock = new Lock("list lock");
          ^
../threads/synchlist.cc:27:15: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
     listEmpty = new Condition("list empty cond");
               ^
g++ -m32 -P -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -Dx86 -DLINUX -c ../threads/switch.S
cc1: note: obsolete option -I- used, please use -iquote instead
g++ bitmap.o debug.o libtest.o sysdep.o interrupt.o stats.o timer.o console.o machine.o mipssim.o translate.o network.o disk.o alarm.o kernel.o main.o scheduler.o synch.o thread.o t
hreadtest.o addrspace.o exception.o synchconsole.o directory.o filehdr.o filesys.o pbitmap.o openfile.o synchdisk.o post.o switch.o -m32 -o nachos
nsuryapr@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -K
*** thread 2 looped 0 times
*** thread 2 looped 1 times
^Z
[1]+  Stopped                 ./nachos -K
nsuryapr@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/halt
Machine halting!

Ticks: total 22, idle 0, system 10, user 12
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
nsuryapr@lcs-vc-cis486:~/nachos/code/build.linux$
```

As previously discussed, when halt is executed the NachOS machine stops execution and comes to a halt where it provides us users with statistics. So, the execution of the program is as follows:

Initially the program enters into main.cc, where the program goes to line 212 and when it encounter -X flag, it sets the userProgName value as halt.c. Following this, the RunUserProg method is invoked and the execution for this method begins. The execution involves dedicating an address space for program execution and then executing the userProgName file. Now halt.c is called and executed. The main method in halt.c calls Halt() function.

The kernel switches from user to kernel mode when Halt method is invoked and also further invokes the interrupt system call where the code is:

```
void
Interrupt::Halt()
{
cout << "Machine halting!\n\n";
kernel->stats->Print();
delete kernel;    // Never returns.
}
```

On understanding the above code, it prints that the machine is halting and its sttistics, also deletes the kernel to shut down the entire machine.

(7) **(10 pts)** What would happen when you execute the shell program (code/test/shell.c)? Explain why it happens.

Answer:

Below is a screenshot where we execute the shell program; in shell.c we encounter 4 system calls – Read, Write,Exec and Join. But none of these system calls are defined and hence no execution occurs and the control is passed over to exception.cc to handle the exception caused. Here, under ExceptionHandler function we pass through 2 switches where the first case is SyscallException and the other is default. The first case is matched, followed by the second switch case which further has three cases - Halt, Add and the other is default. So here, default is the case matched; it is a print statement which when executed – prints "Unexpected System Call" along with the type. The code is below:

```
default:
      cerr << "Unexpected system call " << type << "\n";
      break;
```

Since Write is the first exception to be encountered, the type is 8 according to the system call codes that are defined in syscall.h. Then ASSERTNOTREACHED() gets executed which calls the abort() and stops execution.

```
#define ASSERTNOTREACHED()
\
    {                                \
     cerr << "Assertion failed: line " << __LINE__ << "
file " << __FILE__ << "\n";       \
        Abort();
\
    }
```

Screenshot:

```
^Z
[1]+  Stopped                 ./nachos -K
nsuryapr@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/shell
Unexpected system call 8
Assertion failed: line 108 file ../userprog/exception.cc
Aborted (core dumped)
nsuryapr@lcs-vc-cis486:~/nachos/code/build.linux$ ./nachos -x ../test/shell jhiyb
```

(8) **(5pts)** Which emulated MIPS CPU register contains the system call code?

Answer:

As per the comments and explanation on line 32 in the file exception.cc, **Register r2** is the emulated MIPS CPU register that contains the system call code.

The instruction 'syscall' is only an indication that a system call has been requested and does not indicate which one of the system calls to actually perform. Conventionally, the user programs are known to place the value that indicates the particular system call into MIPS register 2 or r2 before execution of the system call instruction. Also register r2 on return is expected to hold the function return values, including the system call return values.

(9) **(15 pts)** Explain the difference between the threads created in ThreadTest() and the thread created for executing halt (or shell).

Answer:

Threads created in ThreadTest()	Threads created for executing halt (or shell)
Considered as user level threads.	Considered to be kernel level threads.
These are threads used to run user created or user defined programs.	These are threads used to mainly run system calls in the nachos kernel. It makes use of the hardware and usually does not require inputs from user.
User level threads are faster in comparison.	Due to various management overheads, kernel level threads are slow
Created using: Thread *t = new Thread("forked thread");	Created by the OS

References

1) https://www.cs.odu.edu/~cs471/soumya/commands.html
2) https://users.cs.duke.edu/~narten/110/nachos/main/node12.html
3) http://www.cas.mcmaster.ca/~rzheng/course/Nachos_Tutorial/nachos.pdf
4) http://people.cs.pitt.edu/~manas/courses/CS1550/nachos.htm

CIS657 Fall 2018

Assignment Disclosure Form

Assignment #: 1

Name: Nidhi Surya Prakash

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?

If Yes, please give the details.

Yes, a co-students attending the same class.

Malhar Ujawane (suid: 642560698)

Sowmya Padmanabhi

Apurva Kemkar

Namratha Arkalgud

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?

If Yes, please give the details.

References

5) https://www.cs.odu.edu/~cs471/soumya/commands.html
6) https://users.cs.duke.edu/~narten/110/nachos/main/node12.html
7) http://www.cas.mcmaster.ca/~rzheng/course/Nachos_Tutorial/nachos.pdf
8) http://people.cs.pitt.edu/~manas/courses/CS1550/nachos.htm

I assert that, to the best of my knowledge, the information on this sheet is true.

Signature:____Nidhi Surya Prakash_____

Date : 9/14/2018

OS A1

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12