CIS657 Spring 2019

Assignment Disclosure Form

Assignment #: 1

Name: Kuchibhotla Lakshmi Harshini

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

If Yes, please give the details.

No

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.

No

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

Signature:__K.L.Harshini._____ Date : 2/1/2019

CIS 657 Lab 1

Submit your Lab 1 document (doc or pdf) including the disclosure form and answers to the Turnitin link on the blackboard (under Assignments).

Due: Feb. 1 (Friday, end of the day)

Late submission: you have 2^d penalty of late days.

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)?

Ans:

Nachos Thread states are READY, RUNNING, BLOCKED, JUST_CREATED. The thread states are defined in the file: thread.h & line 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? (You may not run this when you do your assignment because your VM account is not created yet. Please logically execute Nachos or build locally for test)

Ans:

When the loop is updated as given, 1 line will be printed as below.

*** thread 1 looped 0 times

→This is because when threadtest.cc is executed, SimpleThread() is called twice from ThreadTest(). SimpleThread(0) and SimpleThread(1) are executed.
When SimpleThread(0) is executed, the parameter which=0, and
for (int num = 0; num < which; num++) becomes for (int num = 0; num < 0; num++)
hence it doesn't execute the for loop and when SimpleThread(1) is executed due to the creation of new thread in line 20, the which parameter equals 1, and

for (int num = 0; num < which; num++) becomes for (int num = 0; num < 1; num++)
So, the printf statement is executed only once.

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

Ans:

We can execute SimpleThread function by setting -K flag. When you run the nachos and set -K flag using command nachos -K, main.cc file takes the control first and sets the threadTestFlag to TRUE in line 216. When the threadTestFlag=1, it invokes the ThreadTest function. Now, threadtest.cc gets the control and executes ThreadTest() which forks the SimpleThread() and calls SimpleThread function.

In ThreadTest(), when a Thread() is called, a new thread is created with state JUST_CREATED. When Fork() is called, the scheduler keeps the thread in ReadyToRun state. Meanwhile, the main thread runs and invokes SimpleThread(0). In SimpleThread(), the kernel makes the current thread Yield() which means the current thread is kept at the back of the queue. Now, newly created thread Simple_Thread(1) is called and executed and again kernel yields this thread.

(4) **(15 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).

Ans:

-d: it enables the users to use the debug flags that are defined in debug.h. Debug flags are to be used via commands in the terminal and this debug function executes only when the correct flag is used.
-rs: It is a random seed which means that a random integer is generated when -rs is passed as a parameter to the kernel object. RandomSlice, a generated random integer call an Alarm class which starts the timer and invokes the callback of this class. It then interrupts the kernel forcing it to yield.
-x: when this flag is set, it just prints the copyright message that is stored as an array in the copyright.h.

Prints: Copyright (c) 1992-1996 The Regents of the University of California. All rights reserved.
-s: forces user programs to be executed in single step.
-x: basically runs a user program. When -x flag is set, it copies the user program to nachos address space and runs it on the nacho process.
-ci: when this flag is set, you need to specify the file for which the console input is given. Stdin is used in default. The input is converted to the synchronized console input which locks and releases the input resources using semaphore.
-co: when this flag is set, specify the file for the output to be kept. Stdout is used in default. It is also locked and released using semaphore.
-n: it uses the mail box and post box implementation to set the network reliability.
-m: it is used to set the machine host id which is used for the network communication.
-K: when this flag is set, it runs a self test of kernel threads to check the working of threads synchronization.
-C: when this flag is set, it uses the synchronous console input and console output to run an interactive console test.
-N: when it is set, it basically runs a two-machine network test i.e; checks the network of one machine with other machine that is running with ids 0 & 1.
-f: when we set this flag, it forces the kernel to format the nachos disk. FileSystem Formatter is used to format the disk.
-cp: used to copy a file from UNIX to Nachos by using the filenames in unix and nachos.
-p: prints all the content of the nachos file to the stdout console. It uses the filename to print the content from.
-r: by mentioning the filename, this removes the specified nachos file from the file system.
-l: lists the contents of the Nachos directory
-D: prints the contents of the entire file system.

(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.

Ans:

The following system calls are currently implemented in nachos.

Halt –

void SysHalt()
{kernel->interrupt->Halt(); } // Halt system call generates an interrupt to the kernel which stops the entire nachos. It also prints the performance stats.

There is no system call to stop the Unix but to print the performance statistics we can use ustat system call in unix. There is a unix command halt but no system call.

Add –

int SysAdd(int op1, int op2)
{return op1+op2; } // It adds the two operands present in the registers r4 and r5, stores the result in r2 and returns the result. Also sets the previous program counter for debugging, increments the pc to pc+4 and set it to the next instruction and sets the next pc NextPCReg for the branch execution.
There is no corresponding system call in Unix so far.

The below are some other system calls which does not have the implementation.
Exit –
void Exit(int status); //It means that this user program is done.
Corresponding Unix System call is exit()
Exec –
SpaceId Exec(char* exec_name); //Run the executable, stored in the Nachos file.
Corresponding Unix System call is execl, execv, execle, execve, execlp, execvp.
Join –
int Join(SpaceId id); // called by one process to wait for the termination of another, referred to by its address space id. When the process id exits, this call returns to allow the caller to progress. Return the exit status.
Create –
int Create(char *name); //Creates a Nachos file but does not open the file.
Remove –
int Remove(char *name); //Removes a Nachos file
Open –
OpenFileId Open(char *name); //Open a Nachos file and returns an "OpenFileId" that can be used to read and write to the file. This doesn't specify any mode like open for reading, open for writing
Corresponding Unix System call is open()
Read –
int Read(char *buffer, int size, OpenFileId id); //Read given size bytes from the open file into buffer.
Corresponding Unix System call is read()
Write –

int Write(char *buffer, int size, OpenFileId id); //Write size bytes of data from buffer to the open file.
Corresponding Unix System call is write()

Seek –

int Seek(int position, OpenFileId id); //Set the seek position of the open file to the byte number.

Close –

int Close(OpenFileId id); //Closes the file it means that we are done reading and writing to it.
Corresponding Unix System call is close()

ThreadFork -

ThreadId ThreadFork(void (*func)()); //Creates a new thread to run the procedure in the same existing address space as the current calling thread.
Corresponding Unix System call is fork()

ThreadYield –

void ThreadYield(); //Yield the CPU to another runnable thread, whether in this address space or not.

ExecV –

SpaceId ExecV(int argc, char* argv[]); //It runs the executable which is stored in the Nachos file "argv[0]", with parameters stored in argv[1..argc-1]
Corresponding Unix System call is execv()

ThreadExit –

void ThreadExit(int ExitCode); //It deletes the current thread and returns the ExitCode of the thread to every waiting local thread.
Corresponding Unix System call is _exit()

ThreadJoin –

int ThreadJoin(ThreadId id); //Blocks current thread until local thread “id” exits with ThreadExit. This function returns the ExitCode of ThreadExit() for the exiting thread.

(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.

Ans:

When you execute the halt program using the given command, we get the following output:
(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)
This is because when halt.c is executed, it invokes Halt() in line 18 which is a system call and exception.cc directly gets the control. This is a SysCallException, so it checks for the SC_Halt system call in switch() and invokes SysHalt where, ksyscall.h gets the control and generates an interrupt then passes the control to interrupt.cc and invokes Halt(). Here, the Print() is invoked in stats.cc where all the performance stats are printed and then the kernel is deleted which means nachos is stopped and cleaned.

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

Ans:

When we execute the shell program using the command → ./nachos -x ../test/shell we get the following output:
(Unexpected system call 8, Assertion failed: line 108 file ../userprog/exception.cc, Aborted (core dumped)).
This is because when shell.c is executed, "Write" system call is invoked in line 17. So, exception.cc directly gets the control as it is a SysCallException and in switch(type) it checks for the type SC_Write system call type. As it cannot find its type in switch(), it goes to the default case and prints the error using standard output stream cerr thereby breaking the loop. Finally ASSERTNOTREACHED() is invoked in line 108 of exception.cc giving control to debug.h to print the assertion failed error and it aborts. The system call code for SC_Write is 8 defined in syscall.h.

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

Ans:

The CPU register r2 contains the system call code.

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

Ans:

When System Calls are invoked, the threads created to execute the system calls are kernel threads. The threads created in ThreadTest() are kernel-level threads, and the threads created for executing the system calls in halt and shell programs are kernel level threads. Kernel threads are those which are created, scheduled and managed by the kernel and User threads are those whose existence is unknown to the kernel. Initially, the halt and shell are executed on user threads, when the system calls like Halt(), Write() are invoked, the kernel threads are created to execute the system calls. Advantages of User level threads is easy management, simple representation and the thread switching is fast and efficient. Advantages of Kernel level threads is: as the kernel threads are scheduled by the kernel/os, it can decide to give more time for a process that have more threads to execute that a process that have less threads. Disadvantages of Kernel threads is they are much slower than user threads, it requires a complete thread control block for each thread to store all its information thereby creating overhead and making it complex.

(10) **(10 pts)** What is the physical frame size of NachOS? How many frames in Main Memory? What is total size of main memory? Where are these defined?

Ans:

The physical frame size = 128 bytes

Number of frames in main memory = 128

Total size of main memory = 128*128 bytes
= 2^7 * 2^7 bytes
= 2^14 bytes
= 2^10 * 2^4 bytes
= 16 KB

All these are defined in the file: machine.h and lines 30, 38, 40