CS 1550 Assignment 2 - NACHOS Survey Answers

General questions

1. Explain relations among three files under the code directory: Makefile, Makefile.common, Makefile.dep. Explain what each file does.
<A>
Makefile: This command compiles the files in each directory in *code* and places an executable file called nachos in it.

Makefile.common: This file is included by all the (Makefiles) in the subdirectories.
For e.g. the Makefile in the threads directory includes this Makefile in the code directory by including the following line

include ../Makefile.common

Makefile.dep: This file is used to specify system-dependent parts of the Makefile environment.
For e.g the Makefile in the threads directory includes this Makefile in the code directory by including the following line

include ../Makefile.dep

</A>
2. Explain what each sub-directory is for under the code directory. They are: bin, filesys, machine, network, test, threads, userprog, and vm.
<A>
Bin:
This contains the following modules.

1. noff
   Data structures defining the Nachos Object Code Format.
2. coff2noff.
   In order to run user programs on Nachos, we first compile them using *gcc* and then convert them from the *coff* format to the *Noff* format. This is done using the *coff2noff* program
3. execute
   Used to execute instructions.
4. opstrings
   This file contains the opcodes for the instructions.

Filesys: This component manages all File system operations.

1. Directory management.
2. Open file management.
3. Synchronous disk access.
4. Disk file header.

Machine: The Nachos Machines folder provides simulation of hardware to execute processes - it simulates the MIPS architecture.

Network: This folder is used to simulate the behavior of a network of workstations, each running by connecting the Unix processes via sockets. Nachos operating system supports networking, which implements a simple "post office" facility. Several independent simulated Nachos machines can talk to each other through a simulated network. Unix sockets are used to simulate network connections among the machines

Test:
User test programs to run on the simulated machine. As indicated earlier, these are separate from the source for the Nachos operating system and workstation simulation. This directory contains its own Makefile. The test programs are very simple and are written in C rather than C++.

Threads:
This folder consists of data structures for managing threads. A thread represents sequential execution of code within a program.
1. Representation of thread, fork, finish, yield and sleep operations
2. Scheduler for threads.
3. Context switch.
4. Synchronization primitives.
5. Synchronized list access

Userprog:
1. This folder manages user address space.
2. Handles system call, exception, interrupt.

Vm:
This folder manages the Virtual memory system for Nachos.
</A>

Nachos time

1. Find the variable responsible for keep tracking the time in Nachos. What is it?
<A>The variable is totalTicks. It can be referenced in the code (stats->totalTicks).
This value is increased every time by incrementing it by TimerTicks.
</A>

2. Explain how Nachos time advances. Is it the same as the real-time?
<A> A hardware timer generates a CPU interrupt every X milliseconds. The Nachos timer emulates a hardware timer by scheduling an interrupt to occur every *TimerTicks*.The variable, which causes an interrupt, is defined in stats.h.
</A>
3. What is the purpose of the private variable “randomize” in class Timer?

<A> Interrupts can be caused to occur at random or at fixed intervals. If set, interrupt occur at random intervals. This means that process switching takes place at random intervals. If *randomize* is not set, interrupts take place at fixed intervals, as defined by *TimerTicks* .
</A>

List Questions

1. Find in Nachos code an example of constructing a list object using the List class defined in list.[cc,h]. Desribe how the object is instantiated, whether it is sorted list or non-sorted list.

<A>
synchlist.h: List* list;
synchlist.cc: list = new List();
</A>

1. Study list.[h,cc].. How is the list sorted by SortedInsert()?

<A> The list is sorted in increasing order of the sort key </A>

2. Write the code (no programming needed) for instantiating a list object that can sort items in the increasing order of integer number using the List class.

<A>
```
Item* item;
List* list;
list = new List();
list.SortedInsert(item,1);
```
</A>

Main

1. What does the –rs flag do?

<A> The -rs command line flag causes Nachos to call Thread::Yield on your behalf at semi-random times. the **-rs** flag causes Nachos to decide whether or not to do a context switch after each and every instruction executes
</A>

2. What does the "-d" flag do? How do we enable all flags (besides writing them all out)?
<A> nachos -d to activate the DEBUG statements in threads and machine .The "-d" flag specifies a list of characters for which debug messages are to be enabled. The flag "-d +" enables all the debug messages.
</A>

3. Explain DEBUG('t', "Entering main") at line 84 of main.cc
Explain #ifdef THREAD ThreadTest(); #endif at line 87 of main.cc
Run nachos under the threads directory. It will print out "*** Thread 0 looped 0 times. *** Thread 1 looped 0 times, etc…." Give names of Thread 0 and Thread 1. I.e., what are Thread 0 and Thread 1?

<A>

- DEBUG('t', "Entering main")
  print DEBUG statements, based on a command line argument.
  't' -- thread system

see threads/utility.h for more details.

- #ifdef THREAD ThreadTest(); #endif
  If THREAD is defined, then invoke that function.
- **Thread0 and Thread1 are running threads (Processes in OS term)**

</A>

Thread questions

1. What are the PCB contents in Nachos?

<A>

It includes the program counter, the processor registers, and the execution stack.

</A>

2. Explain the functionalities of Thread.Yield() and Thread.Sleep(). Differences?

<A>Yield:relinquish control over the CPU to another ready thread
Sleep: relinquish control over the CPU, but thread is now blocked. In other words, it will not run again, until explicitly inserted in the ready queue. </A>

3. How do Thread::RestoreUserState() and AddressSpace::RestoreState() differ?

<A>Thread::RestoreUserState() saves the user-level thread state (i.e. the state of the machines registers). AddressSpace::RestoreState() is used to restore address space-specific state (i.e., the state of the address space associated with the thread).
</A>

Scheduler Questions

What does currentThread->space->RestoreState() do?

<A>It restores any address-space specific machine state. In the code as it exists, it restores the state (location) of the page-table.
</A>

What is the scope of the variable currentThread?

<A>

Global

</A>

What is the name of ready queue in Nachos?

<A>Scheduler</A>

How Scheduler::ReadyToRun and Scheduler::Run differ in functionality?

<A>

Scheduler::ReadyToRun: Thread can be dispatched.
Scheduler::Run: Cause nextThread to start running

</A>

Switch Questions

1. What information do we need to save for a context switch (for the DEC MIPS)?
<A>

a. The stack pointer
b. The callee-save registers
c. The frame pointer
d. The return address of the thread

</A>
2. Explain (in a brief paragraph) what happens in SWITCH().
<A>The register state of the thread being switched out is saved into the thread's data structure. The register state of the thread being switched in is restored from the thread's data structure. Control is transferred to the point in the thread where the newly switched-in thread left off processing (i.e. immediately after the thread's SWITCH call).
</A>

Synch Questions

1. The implementation of semaphore in synch.cc has a while-loop, which is different from the one in the textbook. Why do we need this while loop?
   <A>
   Wait until semaphore value > 0, then decrement. Inside while loop, semaphore not available so go to sleep. The while-loop is needed since a thread is put to the end of of the ready queue. While waiting for it's turn, another thread ahead of the thread in the ready queue can "steal" the semaphore value.
   </A>
2. Explain the following code in synch.cc:

   ```
   IntStatus oldLevel = interrupt->SetLevel(IntOff);
     … some code
   (void) interrupt->SetLevel(oldLevel)
   ```

   <A>
   Turn off status, … , restore the old status.
   </A>
3. Study the constructor and destructor functions for the Semaphore class. A queue is created and deleted in the respective functions. Why do we need a queue? i.e., what is the purpose of the queue? What are the items stored in the queue?
   <A>
   threads waiting in P() for the value to be > 0
   </A>

System Questions

1. Explain the usage of TimerInterruptHandler() by Nachos?
<A>TimerInterruptHandler () is used to force context switches at random, unpredictable (but repeatable) locations.
</A>
2. How can we use TimerInterruptHandler() as an operating system CPU scheduler?

<A> We hook a call to the scheduler in TimerInterruptHandler, so that the scheduler gets called at regular intervals. This would be a round-robin scheduler
</A>
3. What does the "-s" flag do?
<A>The "-s" flag puts the MIPS simulator into debug mode, forcing a call to the debugger after the simulator executes each instruction.
</A>

4.Where are the debug flags finally stored?
<A> The debug flags are stored in the global variable enableFlags </A>

ThreadTest Questions

1. Trace SimpleTest and ThreadTest in threadtest.cc
   <A> </A>
2. Who calls ThreadTest()?
   <A> scheduler </A>
3. What does the Fork function call do?
   <A> create a thread to run a procedure concurrently
   with the caller (this is done in two steps -- first
   allocate the Thread object, then call Fork on it)
   </A>
4. In the line "t->Fork(SimpleThread, 1)", what does the 1 represent?
   <A> InitialArgState</A>
5. How do you instantiate a mutex semaphore in ThreadTest.cc?
   <A>Semaphore sm = Semaphore("ThreadTest", 1);</A>
6. What does Thread::Yield() do?
   <A>relinquish control over the CPU to another ready thread</A>
7. What does Thread::Sleep() do? How is this function different from Thread::Yield()?
   <A>relinquish control over the CPU, but thread is now blocked. In other words, it will not run again, until explicitly put back on the ready queue.
   </A>

Utility Questions

1.What is enableFlags?
<A>The debug flags are stored in the global variable enableFlags.</A>

2. What does ASSERT do if the condition is FALSE?
<A>If the condition is not true (i.e., the expression evaluates to 0), then your program will print a message and crash right there before things get messed up further
</A>