A_3

the Third Assignment

Connor Taffe

February 5, 2015

The following is my report for Assignment 3, a report for Lab 3-1. The following are the enumerated tasks outlined in the assignment.

$\mathbf{T}_1$

First, I started `emacs` and used the M-x prompt to spawn `gdb nachos`. Then, I set a breakpoint at function call `Initialize(argc, argv)`. The following is the output from the shell, `emacs`, and `gdb`.

```
emacs -nw
(emacs overwrites terminal buffer)
M-x gdb
Run gdb (like this): gdb nachos
Current directory is ~/nachos-3.4/code/threads/
... (gdb message omitted)
(gdb) list
... (several lines omitted)
82      main(int argc, char **argv)
83      {
84          int argCount; // the number of arguments
85                        // for a particular command
86
87          DEBUG('t', "Entering main");
88          (void) Initialize(argc, argv);
(gdb) break 88
Breakpoint 1 at 0x8048b5e: file main.cc, line 88.
```

$\mathbf{T}_2$

Then I stepped into `Initialize(argc, argv)`, finished all the statements up to `currentThread = new Thread("main");`.

```
(gdb) step
Initialize (argc=1, argv=0xbfffbfa4) at system.cc:81
(gdb) next
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb) print Scheduler
Attempt to use a type name as an expression
(gdb) print scheduler
$1 = (Scheduler *) 0x804f0c8
(gdb) print threadToBeDestroyed
$2 = (Thread *) 0x0
(gdb) print *scheduler
$3 = {readyList = 0x804f0d8}
(gdb)
```

a. The value of `Scheduler` is a memory address: 0x804f0c8.

b. The value of `threadToBeDestroyed` is null or 0x0.

T_3

Then I finished the next two statements and then answered questions a and b.

```
(gdb) next
(gdb)
(gdb) print currentThread
$4 = (Thread *) 0x804f0e8
(gdb) print *currentThread
$5 = {stackTop = 0x0, machineState = {0 <repeats 18 times>},
  stack = 0x0, status = RUNNING, name = 0x804c54e "main"}
```

a. The value of `currentThread` is the memory address 0x804f0e8, the value of `*currentThread` is the following structure:

```
{ stackTop = 0x0,
  machineState = {0 $<repeats 18 times>},
  stack = 0x0,
  status = RUNNING,
  name = 0x804c54e "main" }
```

b. `currentThread` points to an object of type Thread.

$\mathbf{T}_4$

I then finished the `Initialize(argc, argv);` and returned to `main` by running the following commands in gdb:

```
(gdb) next
(gdb)
main (argc=1, argv=0xbfffd0a4) at main.cc:91
(gdb)
```

$\mathbf{T}_5$

Next, I stepped into the `ThreadTest` function.

```
(gdb) step
ThreadTest () at threadtest.cc:44
(gdb)
```

$\mathbf{T}_6$

Then, I finished the `DEBUG()` and `Thread *t = new Thread("forked thread");` statements.

```
(gdb) next
(gdb) next
(gdb) print t
$1 = (Thread *) 0x804f148
(gdb) print *t
$2 = {stackTop = 0x0, machineState = {0 <repeats 18 times>},
  stack = 0x0, status = JUST_CREATED,
  name = 0x804c64b "forked thread"}
```

a. The value of `t` is a Thread pointer with the value 0x804f148. The value of `*t` is the following structure:

   ```
   { stackTop = 0x0,
     machineState = {0 <repeats 18 times>},
     stack = 0x0,
     status = JUST_CREATED,
     name = 0x804c64b "forked thread" }
   ```

b. The object pointed to by `t` is of type Thread.

T_7

Then I stepped into the function `t->Fork(SimpleThread, 1);`.

```
(gdb) step
Thread::Fork (this=0x804f148, func=0x804a8b8 <SimpleThread(int)>, arg=1)
at thread.cc:95
(gdb)
```

T_8

Then I finished the `DEBUG()` call and stepped into the function call `StackAllocate(func, arg);`.

```
(gdb) next
(gdb) step
Thread::StackAllocate (this=0x804f148, func=0x804a8b8 <SimpleThread(int)>,
arg=1) at thread.cc:260
(gdb)
```

T_9

Then I finished up to the first machineState assignment, `machineState[PCState] = (_int) ThreadRoot;`.

```
(gdb) next
(gdb)
(gdb)
(gdb)
```

T_{10}

The questions a-d are answerd as follows:

```
(gdb) print stackTop
$3 = (int *) 0x8054198
(gdb) print stack
$4 = (int *) 0x80501a8
(gdb) print ThreadRoot
$5 = {<text variable, no debug info>} 0x804c22c <ThreadRoot>
(gdb) print InterruptEnable()
$6 = void
(gdb) print InterruptEnable
$7 = {void (void)} 0x804a31c <InterruptEnable()>
(gdb) print ThreadFinish
```

```
$8 = {void (void)} 0x804a53c <ThreadFinish()>
(gdb) print func
$9 = (VoidFunctionPtr) 0x804a8b8 <SimpleThread(int)>
(gdb) print arg
$10 = 1
(gdb) whatis $
type = int
(gdb)
```

a. The value of `stackTop` is 0x8054198. The value of `stack` is 0x80501a8.

b. The starting addresses of function `ThreadRoot` is 0x804c22c. The starting address of function `InterruptEnable` is 0x804a31c. The starting address of function `ThreadFinish` is 0x804a53c.

c. The value of parameter `func` is 0x804a8b8. `func` is an object of type `VoidFunctionPtr` which points to the function `SimpleThread(int)`.

d. The value of parameter `arg` is 1. `arg` is an object of type `int`.

T_{11}

Then I finished the function `StackAllocate(func, arg);` and returned back to function `t->Fork(SimpleThread, 1);`.

```
(gdb) next
(gdb) next
(gdb) next
(gdb) next
(gdb) next
(gdb)
Thread::Fork (this=0x804f148, func=0x804a8b8 <SimpleThread(int)>, arg=1)
at thread.cc:100
(gdb)
```

T_{12}

I then finished `IntStatus oldLevel = interrupt->SetLevel(IntOff);` and answered the questions in the following:

```
(gdb) print this
$1 = (Thread * const) 0x804f148
(gdb) print *this
$2 = {stackTop = 0x8054198, machineState = {0, 0, 134521628, 1, 0, 134523064,
134522172, 134529580, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, stack = 0x80501a8,
status = JUST_CREATED, name = 0x804c64b "forked thread"}
```

```
(gdb) print /x *this
$3 = {stackTop = 0x8054198, machineState = {0x0, 0x0, 0x804a31c, 0x1, 0x0,
0x804a8b8, 0x804a53c, 0x804c22c, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0}, stack = 0x80501a8, status = 0x0, name = 0x804c64b}
```

a. The value of `this` is 0x804f148. The value of `*this` is the structure as follows:

```
{ stackTop = 0x8054198,
  machineState = {0, 0, 134521628, 1, 0, 134523064,
    134522172, 134529580, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  stack = 0x80501a8,
  status = JUST_CREATED,
  name = 0x804c64b "forked thread" }
```

b. The binary value of `*this` is as follows:

```
{ stackTop = 0x8054198,
  machineState = {0x0, 0x0, 0x804a31c, 0x1, 0x0,
    0x804a8b8, 0x804a53c, 0x804c22c, 0x0, 0x0,
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  stack = 0x80501a8,
  status = 0x0,
  name = 0x804c64b }
```

c. Yes, the pointer points to the same memory address.

d. Yes, `stackTop` has been assigned a non-zero value, and `machineState` contains some data.

e. Yes. The values for sevaral indices of `machineState` have the same value as those functions have. They are stored in `machineState`.

$\mathbf{T}_{13}$

Then I stepped into the function call `scheduler->ReadyToRun(this);`.

```
(gdb) next
(gdb) step
Scheduler::ReadyToRun (this=0x804f0c8, thread=0x804f148) at scheduler.cc:56
```

$\mathbf{T}_{14}$

I then finished the `DEBUG()` call and `thread->setStatus(READY);` and answered the questions in the following:

```
(gdb) next
(gdb) next
(gdb) print readyList
$4 = (List *) 0x804f0d8
(gdb) print *readyList
$5 = {first = 0x0, last = 0x0}
(gdb)
```

a. The value of `readyList` is 0x804f0d8. The value of `*readyList` is the structure as follows:

   ```
   { first = 0x0,
     last = 0x0 }
   ```

b. Yes, it has not been filled yet as we just set the status our status to 'READY' and have not appended anything to the list yet.

T_{15}

Next, I finished the function call `readyList->Append((void *)thread);` and answered the questions as follows:

```
(gdb) next
(gdb) print readyList
$6 = (List *) 0x804f0d8
(gdb) print *readyList
$7 = {first = 0x80551b0, last = 0x80551b0}
(gdb) print readyList->first
$8 = (ListElement *) 0x80551b0
(gdb) print *readyList->first
$9 = {next = 0x0, key = 0, item = 0x804f148}
(gdb)
```

a. The value of `readyList` is 0x804f0d8. The value of `*readyList` is the following structure:

   ```
   { first = 0x80551b0,
     last = 0x80551b0 }
   ```

b. No, it has one item as `first` and `last` are pointing to the same item, but it is an item and not null.

c. Its first `ListElement` is the following structure:

   ```
   { next = 0x0,
     key = 0,
     item = 0x804f148 }
   ```

   Yes. The value of the item is a pointer to a Thread object, the same thread object in `t` from `ThreadTest`, and `this` in `t->Fork`.

$\mathbf{T}_{16}$

I then finished `scheduler->ReadyToRun(this);` and return back to `t->Fork(SimpleThread, 1);`.

```
(gdb) next
Thread::Fork (this=0x804f148, func=0x804a8b8 <SimpleThread(int)>, arg=1)
at thread.cc:103
(gdb)
```

$\mathbf{T}_{17}$

I then finished `t->Fork(SimpleThread, 1);` and return back to `ThreadTest()`.

```
(gdb) next
(gdb)
ThreadTest () at threadtest.cc:49
(gdb)
```

$\mathbf{T}_{18}$

I then finished the function call `SimpleThread(0);` and reported the output.

```
(gdb) next
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
(gdb)
```

$\mathbf{T}_{19}$

I then finished the `ThreadTest();` function and return to `main(int argc, char **argv)`.

```
(gdb) next
main (argc=1, argv=0xbfffd364) at main.cc:97
(gdb)
```

$\mathbf{T}_{20}$

Next, I finished the `for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount)` loop.

```
(gdb) next
(gdb)
```

$\mathbf{T}_{21}$

I then stepped into function call `currentThread->Finish();` and finished all the statements up to `Sleep()` and answer the questions as following:

```
(gdb) step
Thread::Finish (this=0x804f0e8) at thread.cc:151
(gdb) next
(gdb)
(gdb)
(gdb)
(gdb) print threadToBeDestroyed
$10 = (Thread *) 0x804f0e8
(gdb) print *threadToBeDestroyed
$11 = {stackTop = 0xbfffd1bc, machineState = {134541544, 134530635, 6565120,
724249387, -1073753624, 3415200, 0, 134516763, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0}, stack = 0x0, status = RUNNING, name = 0x804c54e "main"}
(gdb)
```

a. The value of `threadToBeDestroyed` is 0x804f0e8. The value of `*threadToBeDestroyed` is the following structure:

```
{ stackTop = 0xbfffd1bc,
  machineState = {134541544, 134530635, 6565120,
    724249387, -1073753624, 3415200, 0, 134516763, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0},
  stack = 0x0,
  status = RUNNING,
  name = 0x804c54e "main" }
```

b. The object pointed to by `threadToBeDestroyed` is of type Thread. Yes, it points to the same memory address.

c. The stackTop is no longer null and pointed at 0x0, the machineState has data and is not empty.

$\mathbf{T}_{22}$

I then stepped into function call `Sleep();` and finished all the statements up to `scheduler->Run(nextThread);` and answer the questions as following:

```
(gdb) step
Thread::Sleep (this=0x804f0e8) at thread.cc:221
(gdb) next
(gdb)
(gdb)
(gdb)
(gdb)
(gdb) print nextThread
$12 = (Thread *) 0x804f148
(gdb) print *nextThread
$13 = {stackTop = 0x8054100, machineState = {134541640, 134530382, 6565120,
724249387, 134562092, 134523064, 134522172, 134516763, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0}, stack = 0x80501a8, status = READY,
name = 0x804c64b "forked thread"}
(gdb)
```

a. The value of `nextThread` is 0x804f148. The value of `*nextThread` is the following structure:

```
{ stackTop = 0x8054100,
  machineState = {134541640, 134530382, 6565120,
    724249387, 134562092, 134523064, 134522172, 134516763,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  stack = 0x80501a8,
  status = READY,
  name = 0x804c64b "forked thread" }
```

b. The object pointed to by `nextThread` is of type Thread.

$\mathbf{T}_{23}$

I then finished the `scheduler->Run(nextThread);` and answered the questions as follows:

```
(gdb) next
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 130, idle 0, system 130, user 0
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

Cleaning up...

Program exited normally.
(gdb) next
The program is not being run.
(gdb)
```

a. Output of this function is recorded above.

b. No, `scheduler->Run(nextThread);` did not return. No, the program has terminated.