CZ2005: Operating Systems Lab 2

⚠

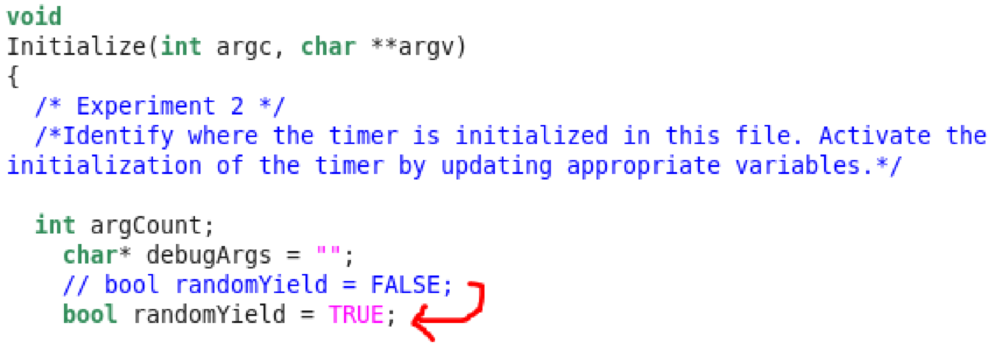
Lab Implementation Guide

(refer to CZ2005_OS_LAB_2.pdf for more details)

Part 1 & 2

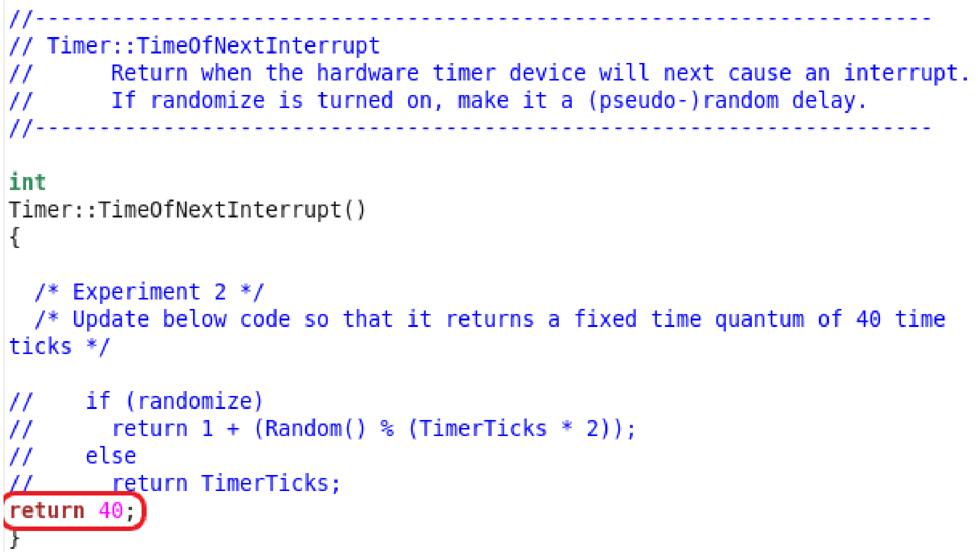
1. Initialize the timer interrupt with a fixed time quantum of **40 time ticks.**

a) Activate Timer in system.cc.



```
void
Initialize(int argc, char **argv)
{
  /* Experiment 2 */
  /*Identify where the timer is initialized in this file. Activate the
initialization of the timer by updating appropriate variables.*/

  int argCount;
    char* debugArgs = "";
    // bool randomYield = FALSE;
    bool randomYield = TRUE;
```

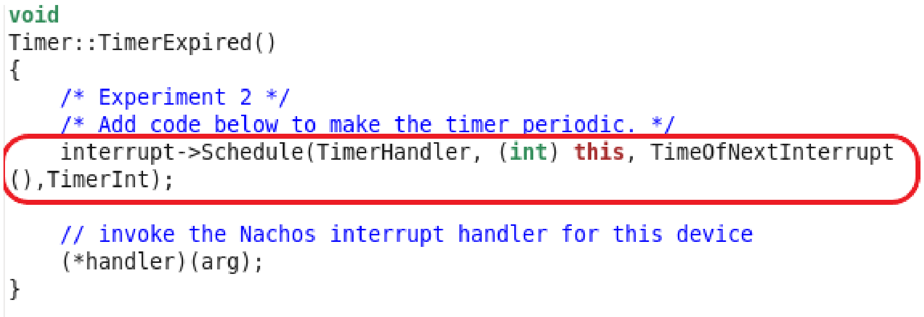
b) Initialize the timer with the fixed time quantum in timer.cc.



```
//----------------------------------------------------------------------
// Timer::TimeOfNextInterrupt
//      Return when the hardware timer device will next cause an interrupt.
//      If randomize is turned on, make it a (pseudo-)random delay.
//----------------------------------------------------------------------

int
Timer::TimeOfNextInterrupt()
{

  /* Experiment 2 */
  /* Update below code so that it returns a fixed time quantum of 40 time
ticks */

//    if (randomize)
//      return 1 + (Random() % (TimerTicks * 2));
//    else
//      return TimerTicks;
return 40;
}
```

2. Make the timer interrupt periodic.

a) Modify function TimerExpired()in timer.cc to make the timer periodic. It should trigger an interrupt every 40 time ticks.



```
void
Timer::TimerExpired()
{
    /* Experiment 2 */
    /* Add code below to make the timer periodic. */
    interrupt->Schedule(TimerHandler, (int) this, TimeOfNextInterrupt
(),TimerInt);

    // invoke the Nachos interrupt handler for this device
    (*handler)(arg);
}
```

c) **In Console:**

>> cd ~/nachos-exp1-2/exp2.

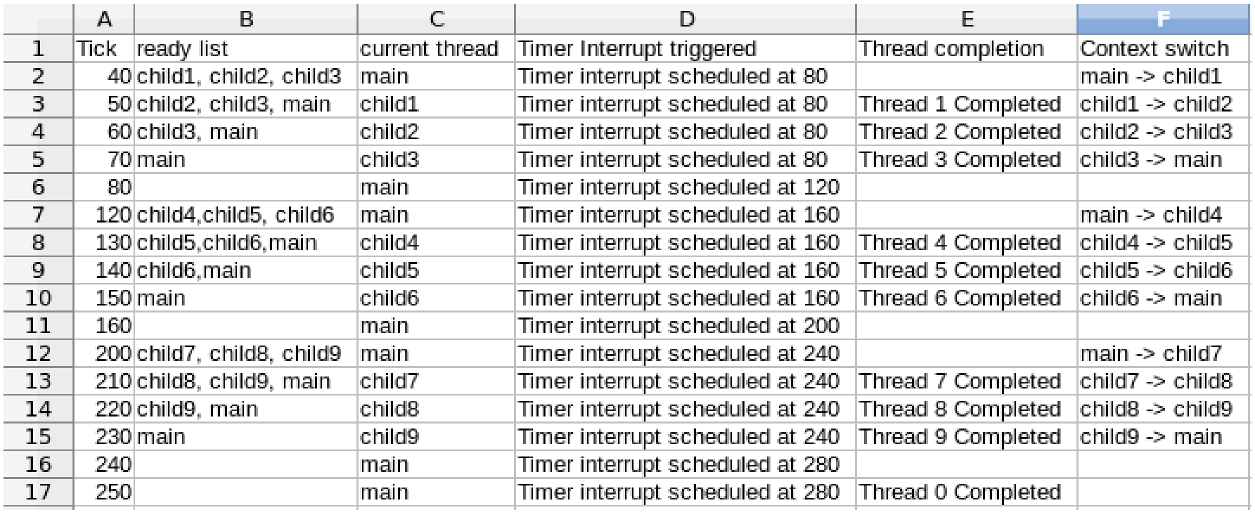
>> make. (Should see "ln -sf arch/intel-i386-linux/bin/nachos nachos")

Else type >> make clean

>> ./nachos -d > output_1.txt. Option –d is to display Nachos debugging messages.

e) Fill in Table1.csv (template is provided)

FINAL TABLE1.CSV



	A	B	C	D	E	F
1	Tick	ready list	current thread	Timer Interrupt triggered	Thread completion	Context switch
2	40	child1, child2, child3	main	Timer interrupt scheduled at 80		main -> child1
3	50	child2, child3, main	child1	Timer interrupt scheduled at 80	Thread 1 Completed	child1 -> child2
4	60	child3, main	child2	Timer interrupt scheduled at 80	Thread 2 Completed	child2 -> child3
5	70	main	child3	Timer interrupt scheduled at 80	Thread 3 Completed	child3 -> main
6	80		main	Timer interrupt scheduled at 120		
7	120	child4,child5, child6	main	Timer interrupt scheduled at 160		main -> child4
8	130	child5,child6,main	child4	Timer interrupt scheduled at 160	Thread 4 Completed	child4 -> child5
9	140	child6,main	child5	Timer interrupt scheduled at 160	Thread 5 Completed	child5 -> child6
10	150	main	child6	Timer interrupt scheduled at 160	Thread 6 Completed	child6 -> main
11	160		main	Timer interrupt scheduled at 200		
12	200	child7, child8, child9	main	Timer interrupt scheduled at 240		main -> child7
13	210	child8, child9, main	child7	Timer interrupt scheduled at 240	Thread 7 Completed	child7 -> child8
14	220	child9, main	child8	Timer interrupt scheduled at 240	Thread 8 Completed	child8 -> child9
15	230	main	child9	Timer interrupt scheduled at 240	Thread 9 Completed	child9 -> main
16	240		main	Timer interrupt scheduled at 280		
17	250		main	Timer interrupt scheduled at 280	Thread 0 Completed	

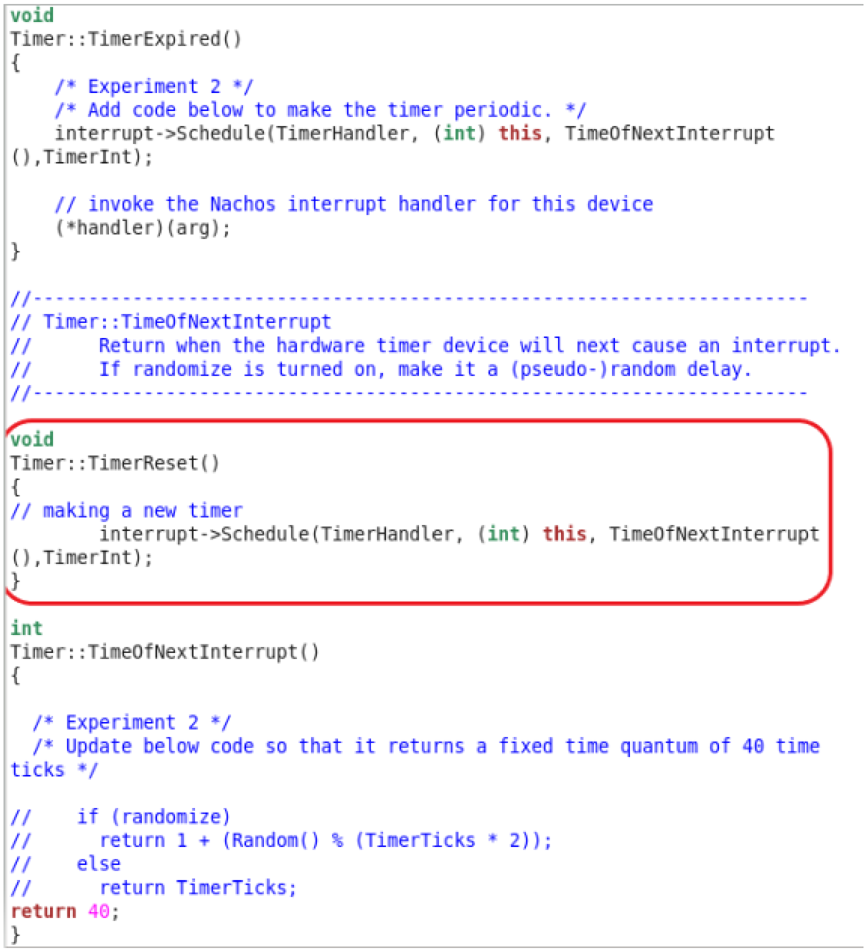
Part 3

3. Reset the timer interrupt if a thread finishes in the middle of a time quantum.

a) When the current thread finishes, remove the pending timer interrupt from the pending list, and insert a new timer interrupt with the time quantum of 40 time ticks. Note: For this experiment, to keep things simple, we will assume that no other interrupts are pending in the list, except the timer interrupts created by us.

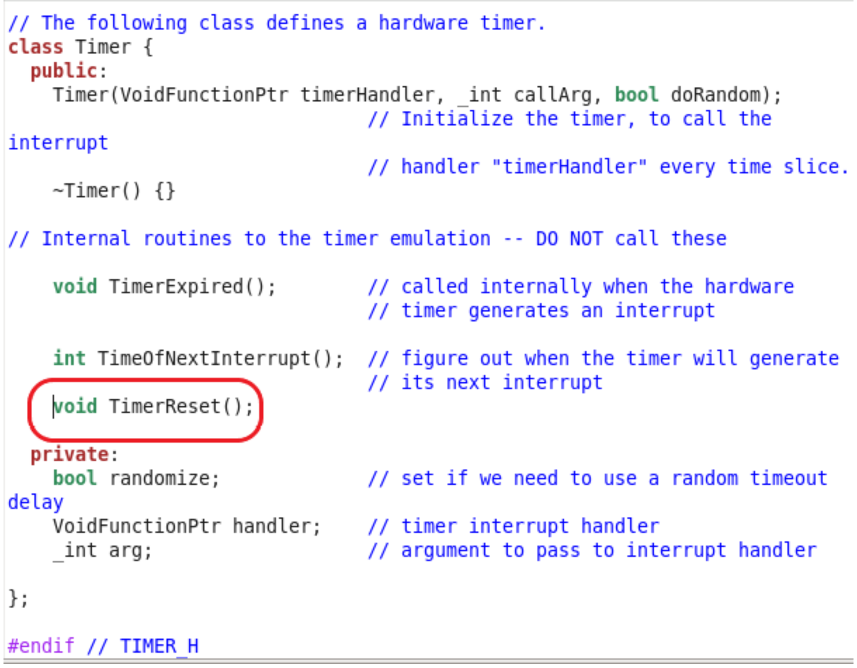
b) To accomplish the above task, you would need to modify files/functions ThreadsFinish(), timer.cc, timer.h, interrupt.cc and interrupt.h.

Timer.cc



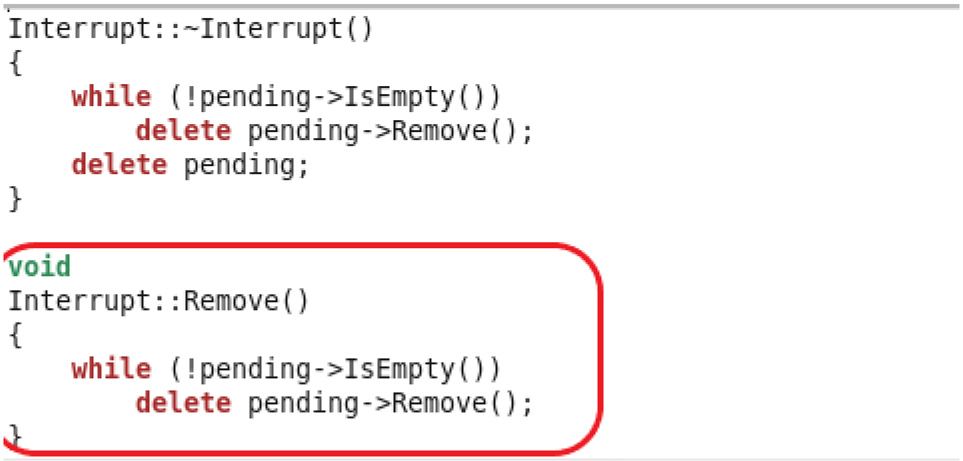
```
void
Timer::TimerExpired()
{
    /* Experiment 2 */
    /* Add code below to make the timer periodic. */
    interrupt->Schedule(TimerHandler, (int) this, TimeOfNextInterrupt
(),TimerInt);

    // invoke the Nachos interrupt handler for this device
    (*handler)(arg);
}

//----------------------------------------------------------------------
// Timer::TimeOfNextInterrupt
//      Return when the hardware timer device will next cause an interrupt.
//      If randomize is turned on, make it a (pseudo-)random delay.
//----------------------------------------------------------------------

void
Timer::TimerReset()
{
// making a new timer
        interrupt->Schedule(TimerHandler, (int) this, TimeOfNextInterrupt
(),TimerInt);
}

int
Timer::TimeOfNextInterrupt()
{

  /* Experiment 2 */
  /* Update below code so that it returns a fixed time quantum of 40 time
ticks */

//     if (randomize)
//       return 1 + (Random() % (TimerTicks * 2));
//     else
//       return TimerTicks;
return 40;
}
```

timer.h



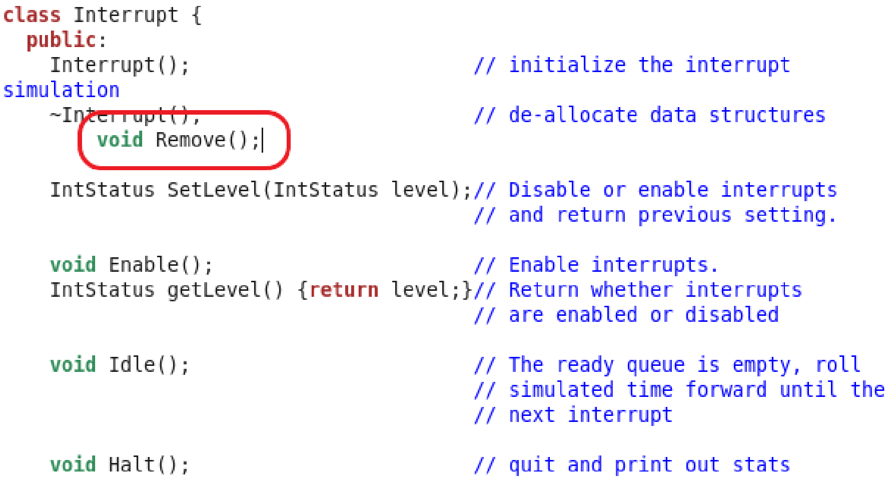
```
// The following class defines a hardware timer.
class Timer {
  public:
    Timer(VoidFunctionPtr timerHandler, _int callArg, bool doRandom);
                                // Initialize the timer, to call the
interrupt
                                // handler "timerHandler" every time slice.
    ~Timer() {}

// Internal routines to the timer emulation -- DO NOT call these

    void TimerExpired();        // called internally when the hardware
                                // timer generates an interrupt

    int TimeOfNextInterrupt();  // figure out when the timer will generate
                                // its next interrupt
    void TimerReset();

  private:
    bool randomize;             // set if we need to use a random timeout
delay
    VoidFunctionPtr handler;    // timer interrupt handler
    _int arg;                   // argument to pass to interrupt handler

};

#endif // TIMER_H
```

interrupt.cc



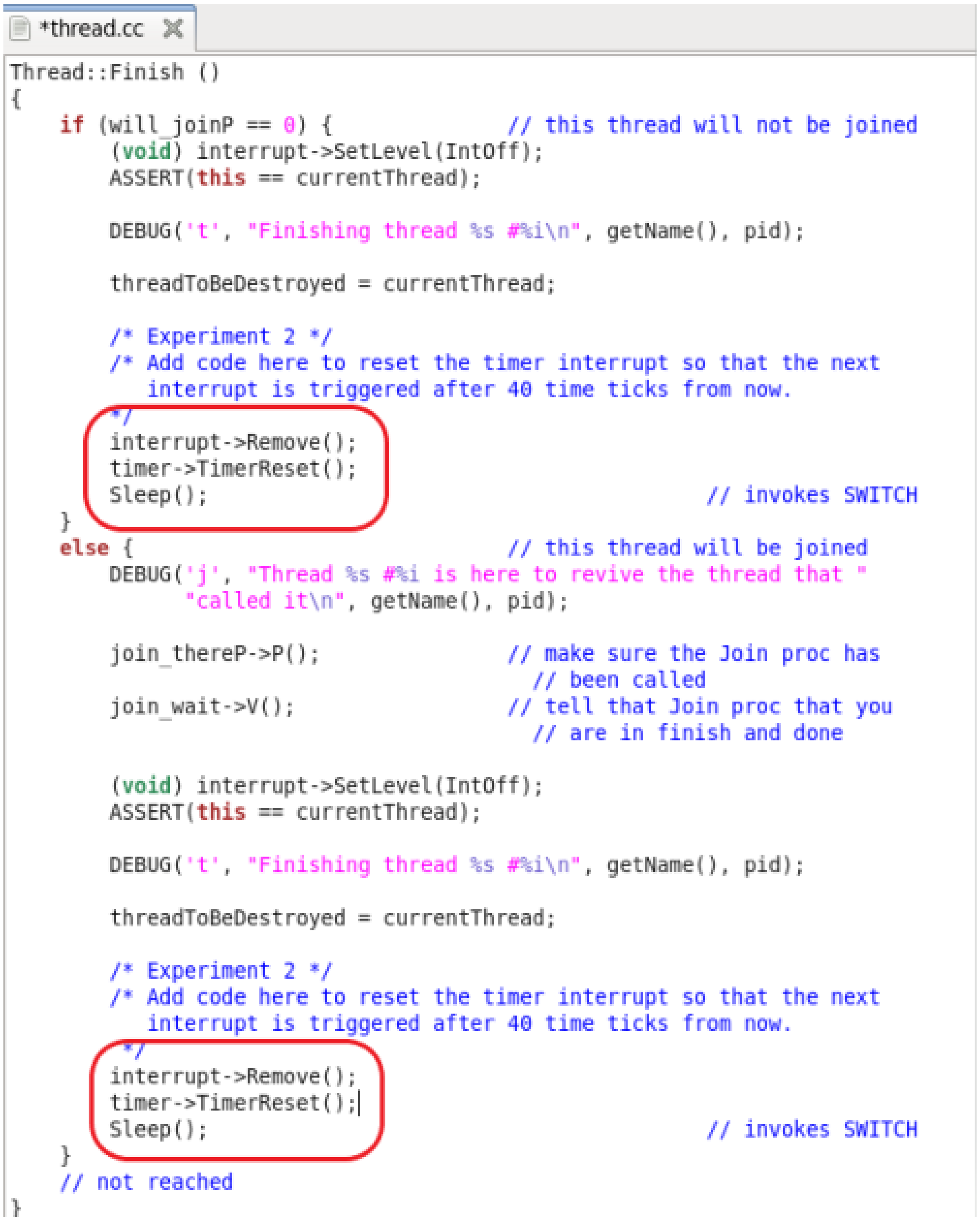
```
Interrupt::~Interrupt()
{
    while (!pending->IsEmpty())
        delete pending->Remove();
    delete pending;
}

void
Interrupt::Remove()
{
    while (!pending->IsEmpty())
        delete pending->Remove();
}
```

interrupt.h



```
class Interrupt {
  public:
    Interrupt();                        // initialize the interrupt
simulation
    ~Interrupt();                       // de-allocate data structures
        void Remove();

    IntStatus SetLevel(IntStatus level);// Disable or enable interrupts
                                        // and return previous setting.

    void Enable();                      // Enable interrupts.
    IntStatus getLevel() {return level;}// Return whether interrupts
                                        // are enabled or disabled

    void Idle();                        // The ready queue is empty, roll
                                        // simulated time forward until the
                                        // next interrupt

    void Halt();                        // quit and print out stats
```

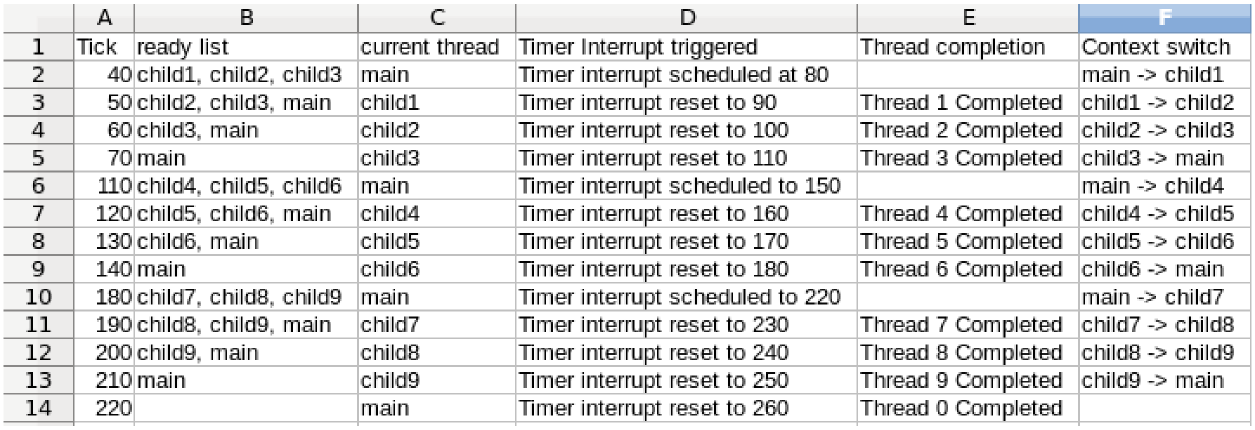
thread.cc



```
Thread::Finish ()
{
    if (will_joinP == 0) {                  // this thread will not be joined
        (void) interrupt->SetLevel(IntOff);
        ASSERT(this == currentThread);

        DEBUG('t', "Finishing thread %s #%i\n", getName(), pid);

        threadToBeDestroyed = currentThread;

        /* Experiment 2 */
        /* Add code here to reset the timer interrupt so that the next
           interrupt is triggered after 40 time ticks from now.
        */
        interrupt->Remove();
        timer->TimerReset();
        Sleep();                                            // invokes SWITCH
    }
    else {                                  // this thread will be joined
        DEBUG('j', "Thread %s #%i is here to revive the thread that "
              "called it\n", getName(), pid);

        join_thereP->P();                   // make sure the Join proc has
                                              // been called
        join_wait->V();                     // tell that Join proc that you
                                              // are in finish and done

        (void) interrupt->SetLevel(IntOff);
        ASSERT(this == currentThread);

        DEBUG('t', "Finishing thread %s #%i\n", getName(), pid);

        threadToBeDestroyed = currentThread;

        /* Experiment 2 */
        /* Add code here to reset the timer interrupt so that the next
           interrupt is triggered after 40 time ticks from now.
        */
        interrupt->Remove();
        timer->TimerReset();
        Sleep();                                            // invokes SWITCH
    }
    // not reached
}
```

c) Compile and execute Nachos as in Step 2 above (use filename output_2.txt to store your results), and fill Table2.csv (template is provided)

FINAL TABLE2.CSV



	A	B	C	D	E	F
1	Tick	ready list	current thread	Timer Interrupt triggered	Thread completion	Context switch
2	40	child1, child2, child3	main	Timer interrupt scheduled at 80		main -> child1
3	50	child2, child3, main	child1	Timer interrupt reset to 90	Thread 1 Completed	child1 -> child2
4	60	child3, main	child2	Timer interrupt reset to 100	Thread 2 Completed	child2 -> child3
5	70	main	child3	Timer interrupt reset to 110	Thread 3 Completed	child3 -> main
6	110	child4, child5, child6	main	Timer interrupt scheduled to 150		main -> child4
7	120	child5, child6, main	child4	Timer interrupt reset to 160	Thread 4 Completed	child4 -> child5
8	130	child6, main	child5	Timer interrupt reset to 170	Thread 5 Completed	child5 -> child6
9	140	main	child6	Timer interrupt reset to 180	Thread 6 Completed	child6 -> main
10	180	child7, child8, child9	main	Timer interrupt scheduled to 220		main -> child7
11	190	child8, child9, main	child7	Timer interrupt reset to 230	Thread 7 Completed	child7 -> child8
12	200	child9, main	child8	Timer interrupt reset to 240	Thread 8 Completed	child8 -> child9
13	210	main	child9	Timer interrupt reset to 250	Thread 9 Completed	child9 -> main
14	220		main	Timer interrupt reset to 260	Thread 0 Completed	