Question 1 (6 points) What is the value of TimerTicks? How often does a timer interrupt occur?

The TimerTicks is defined in machine/stats.h, whose value is 100. The average time of timer interrupt occurs is 100.

Question 2 (6 points) In line 99 of kernel.cc, a timer device alarm is created by the code alarm = new Alarm(randomSlice);. What is the data type of the variable randomSlice and what is its value in this line?

randomSlice is a bool value, which is defined in kernel.h.

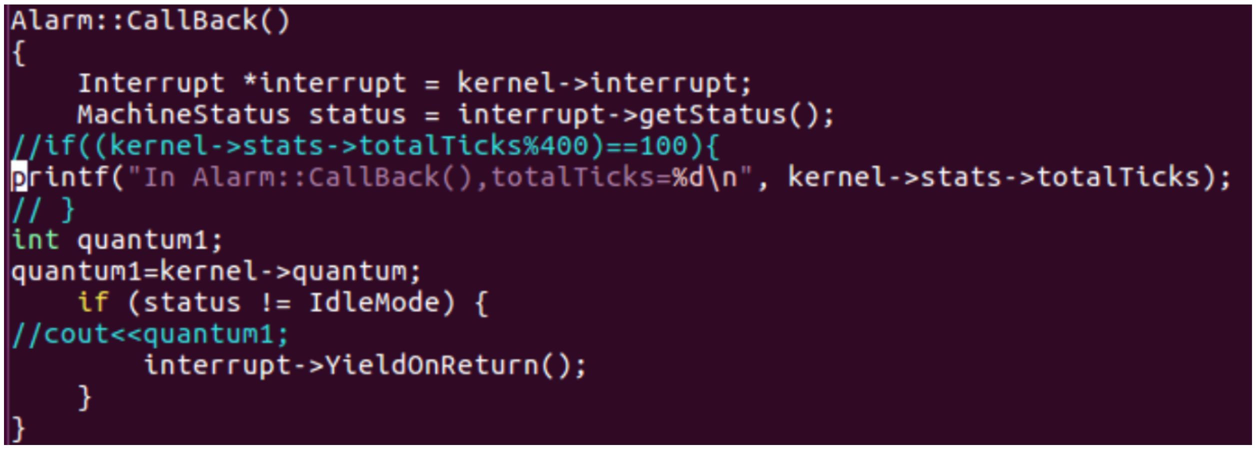
False

Question 3 (6 points) Which source file in the machine directory implements the function OneTick()? What is the function's role? Two situations can cause OneTick() to be called. One of them is when a user program instruction is executed. Which function in which file contains this call when a user instruction is executed?

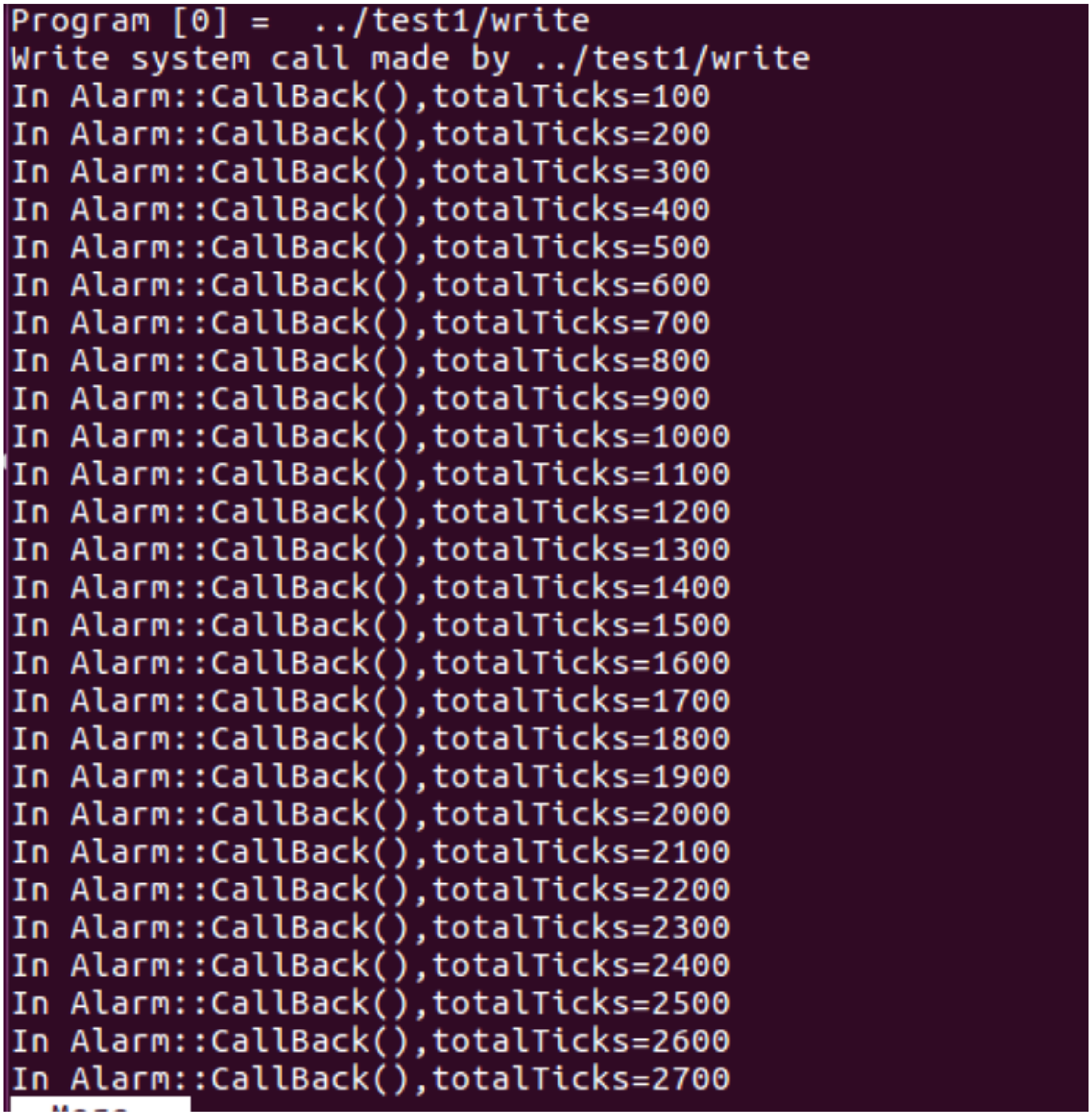
The function OneTick() is implemented in Interrupt.cc. And the function is to advance simulated time and check if there are any pending interrupts to be called.
Will execute
stats->totalTicks+=UserTick;
stats->userTicks+=UserTick;

Question 4 (12 points) Make sure that the timer object alarm is created in line 99 of kernel.cc. Modify void Alarm::CallBack() function in alarm.cc as below by inserting printf("In Alarm::CallBack(), totalTicks = %d\n", kernel->stats->totalTicks);

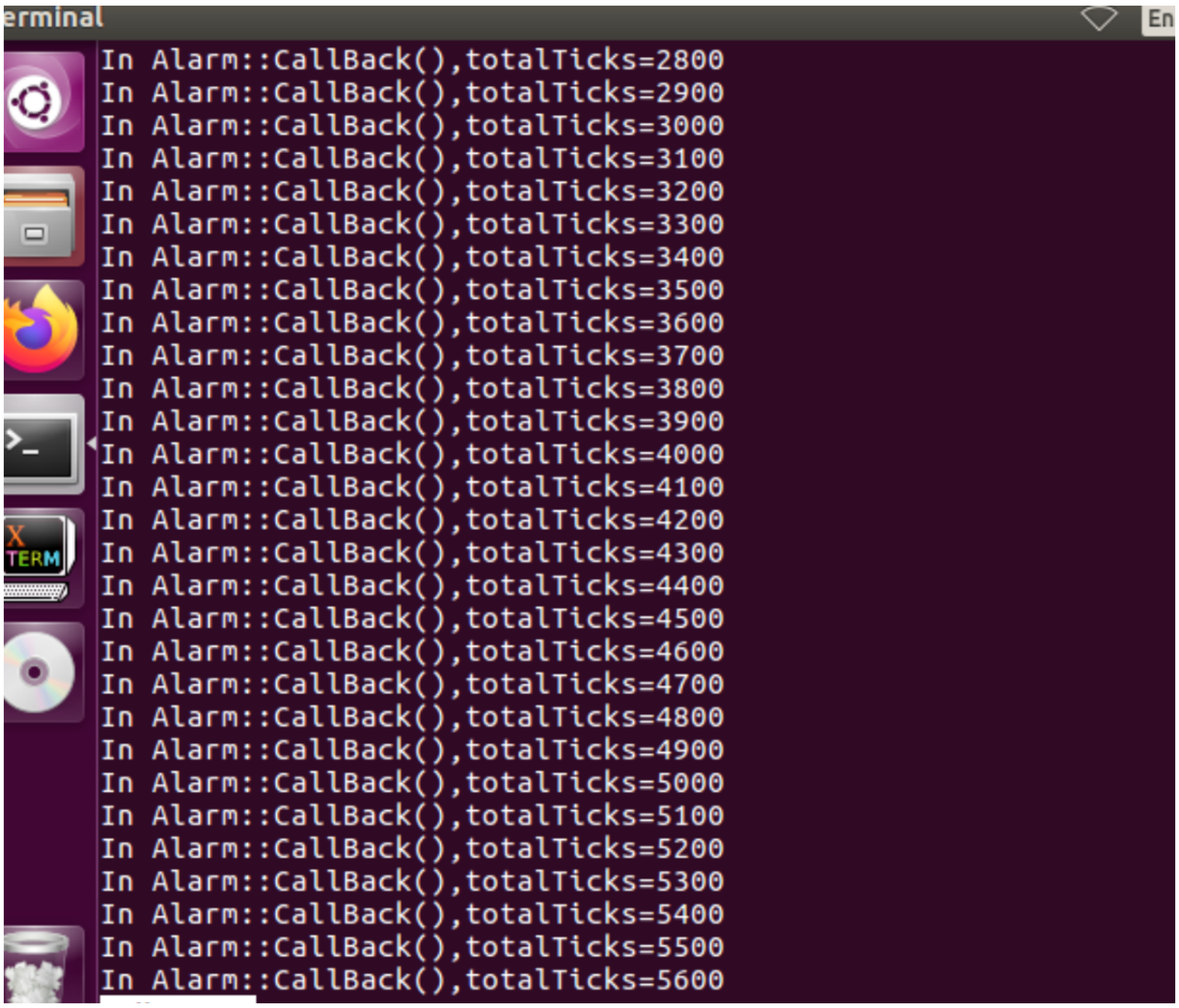
The following screenshot shows modified alarm.cc. And the output of the ./nachos -x ../test1/write | more.



```
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
//if((kernel->stats->totalTicks%400)==100){
printf("In Alarm::CallBack(),totalTicks=%d\n", kernel->stats->totalTicks);
// }
int quantum1;
quantum1=kernel->quantum;
    if (status != IdleMode) {
//cout<<quantum1;
        interrupt->YieldOnReturn();
    }
}
```



```
Program [0] =  ../test1/write
Write system call made by ../test1/write
In Alarm::CallBack(),totalTicks=100
In Alarm::CallBack(),totalTicks=200
In Alarm::CallBack(),totalTicks=300
In Alarm::CallBack(),totalTicks=400
In Alarm::CallBack(),totalTicks=500
In Alarm::CallBack(),totalTicks=600
In Alarm::CallBack(),totalTicks=700
In Alarm::CallBack(),totalTicks=800
In Alarm::CallBack(),totalTicks=900
In Alarm::CallBack(),totalTicks=1000
In Alarm::CallBack(),totalTicks=1100
In Alarm::CallBack(),totalTicks=1200
In Alarm::CallBack(),totalTicks=1300
In Alarm::CallBack(),totalTicks=1400
In Alarm::CallBack(),totalTicks=1500
In Alarm::CallBack(),totalTicks=1600
In Alarm::CallBack(),totalTicks=1700
In Alarm::CallBack(),totalTicks=1800
In Alarm::CallBack(),totalTicks=1900
In Alarm::CallBack(),totalTicks=2000
In Alarm::CallBack(),totalTicks=2100
In Alarm::CallBack(),totalTicks=2200
In Alarm::CallBack(),totalTicks=2300
In Alarm::CallBack(),totalTicks=2400
In Alarm::CallBack(),totalTicks=2500
In Alarm::CallBack(),totalTicks=2600
In Alarm::CallBack(),totalTicks=2700
```



```
In Alarm::CallBack(),totalTicks=2800
In Alarm::CallBack(),totalTicks=2900
In Alarm::CallBack(),totalTicks=3000
In Alarm::CallBack(),totalTicks=3100
In Alarm::CallBack(),totalTicks=3200
In Alarm::CallBack(),totalTicks=3300
In Alarm::CallBack(),totalTicks=3400
In Alarm::CallBack(),totalTicks=3500
In Alarm::CallBack(),totalTicks=3600
In Alarm::CallBack(),totalTicks=3700
In Alarm::CallBack(),totalTicks=3800
In Alarm::CallBack(),totalTicks=3900
In Alarm::CallBack(),totalTicks=4000
In Alarm::CallBack(),totalTicks=4100
In Alarm::CallBack(),totalTicks=4200
In Alarm::CallBack(),totalTicks=4300
In Alarm::CallBack(),totalTicks=4400
In Alarm::CallBack(),totalTicks=4500
In Alarm::CallBack(),totalTicks=4600
In Alarm::CallBack(),totalTicks=4700
In Alarm::CallBack(),totalTicks=4800
In Alarm::CallBack(),totalTicks=4900
In Alarm::CallBack(),totalTicks=5000
In Alarm::CallBack(),totalTicks=5100
In Alarm::CallBack(),totalTicks=5200
In Alarm::CallBack(),totalTicks=5300
In Alarm::CallBack(),totalTicks=5400
In Alarm::CallBack(),totalTicks=5500
In Alarm::CallBack(),totalTicks=5600
```

printf("In Alarm::CallBack(), totalTicks = %d\n", kernel->stats->totalTicks) will output the value of totalTicks when they execute the Alarm::CallBack() function. And the TimerTicks=100, so it will interrupt every 100 ticks. So the output will be multiple of 100.

CallBack() is a software interrupt handler for the timer device. And it is set up to interrupt the CPU periodically(every TimerTicks). And it is called each time there is a timer interrupt, with interrupts diabled. When the write program is running, the timer interrupt will occur every TimerTicks=100, and output the printf("In Alarm::CallBack(), totalTicks = %d\n", kernel->stats->totalTicks);. So the totalTicks is the multiple of 100.

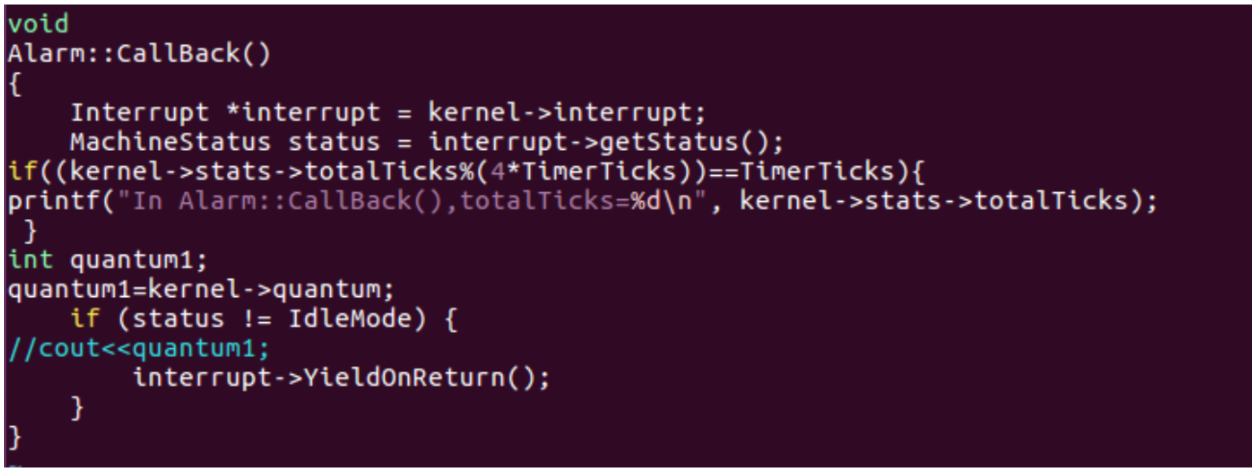
Question 5 (20 points total) In the above question, the “In Alarm::CallBack(), totalTicks = x” message is printed out at every regular interval x.

1. (4 points) What is the value of x?
2. (4 points) Which Nachos source file defines this value of x?

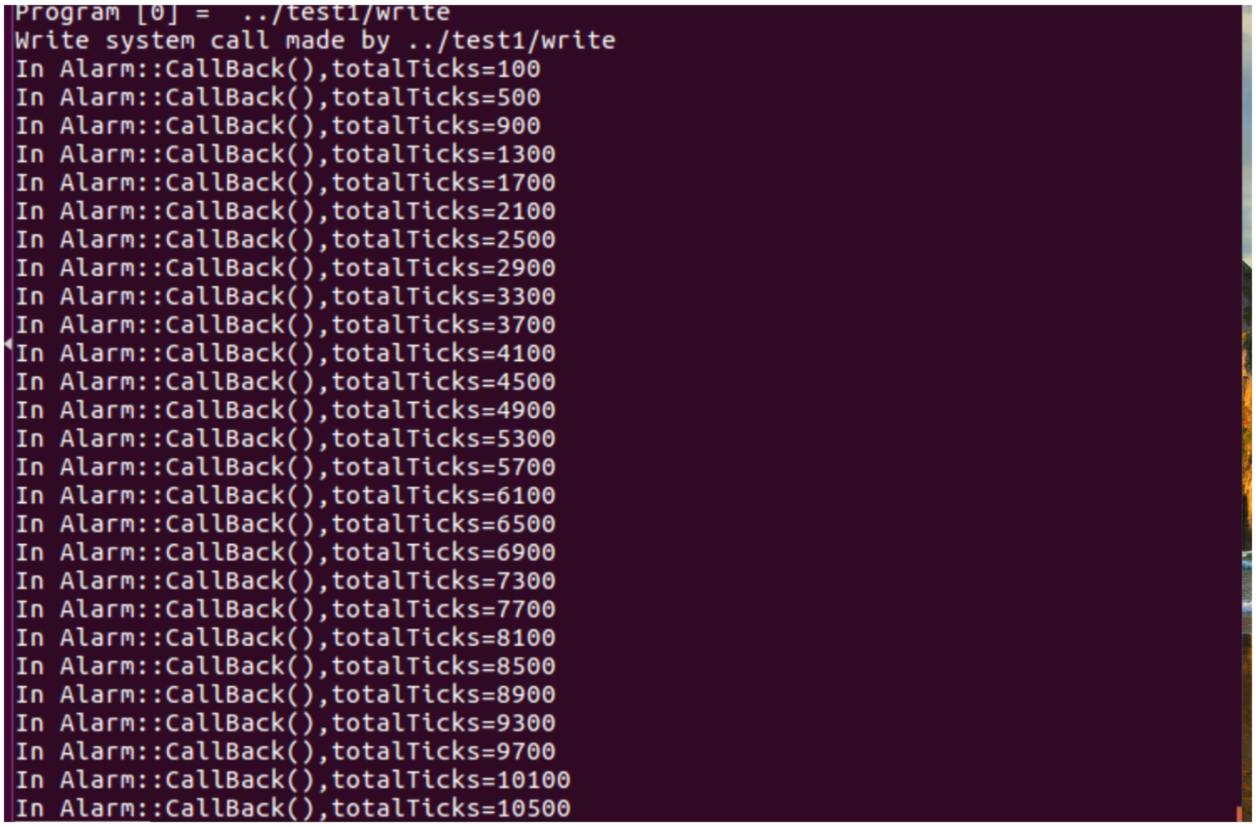
3. (12 points) How should we modify the void Alarm::CallBack() function, if we want the message to be printed out at every 4 × x interval? Give a pseudo code.

1. x=100
2. TimerTicks is defined in nachos/machine/stats.h.
3. The following is the code

Due to the const int TimerTicks=100, so if we want to message to be printed out at every 4*x, just needs to compare totalTicks%(4*TimerTicks)==TimerTicks. The output will meet the requirements.



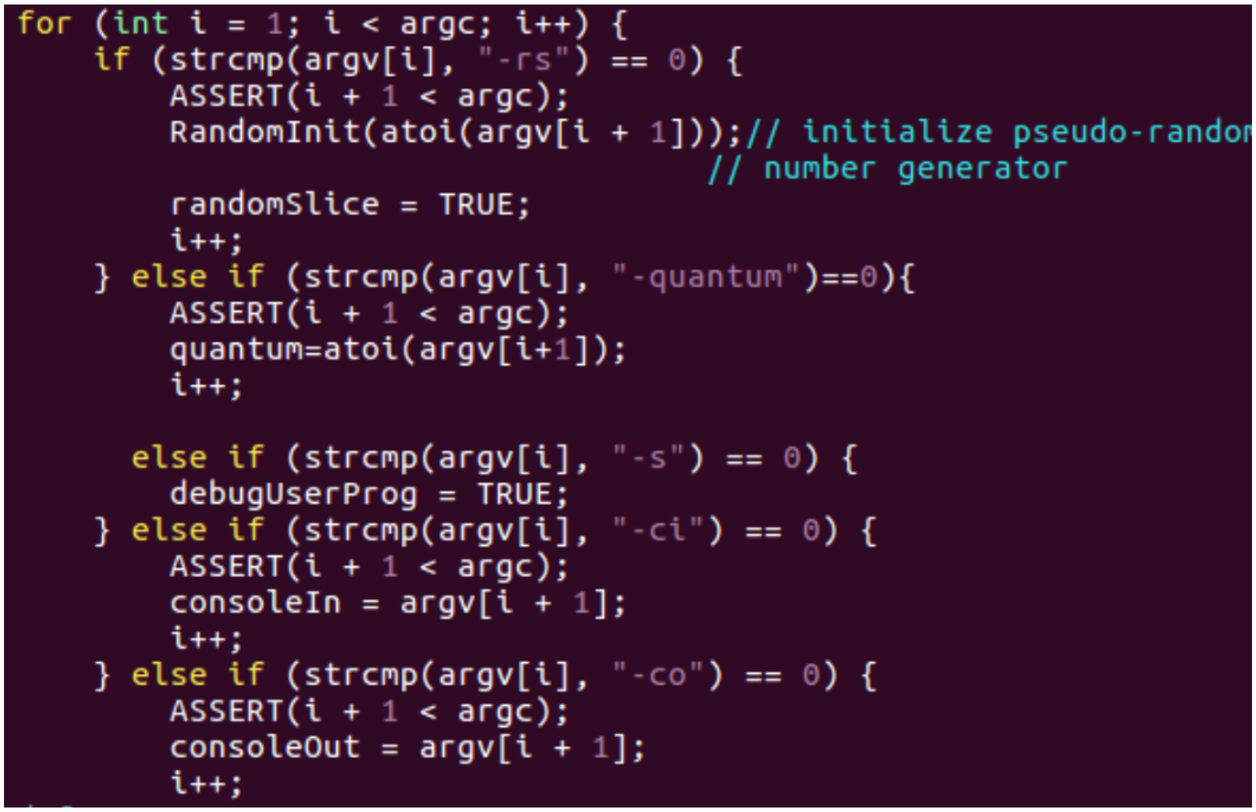
```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
if((kernel->stats->totalTicks%(4*TimerTicks))==TimerTicks){
printf("In Alarm::CallBack(),totalTicks=%d\n", kernel->stats->totalTicks);
 }
int quantum1;
quantum1=kernel->quantum;
    if (status != IdleMode) {
//cout<<quantum1;
        interrupt->YieldOnReturn();
    }
}
```



```
Program [0] =  ../test1/write
Write system call made by ../test1/write
In Alarm::CallBack(),totalTicks=100
In Alarm::CallBack(),totalTicks=500
In Alarm::CallBack(),totalTicks=900
In Alarm::CallBack(),totalTicks=1300
In Alarm::CallBack(),totalTicks=1700
In Alarm::CallBack(),totalTicks=2100
In Alarm::CallBack(),totalTicks=2500
In Alarm::CallBack(),totalTicks=2900
In Alarm::CallBack(),totalTicks=3300
In Alarm::CallBack(),totalTicks=3700
In Alarm::CallBack(),totalTicks=4100
In Alarm::CallBack(),totalTicks=4500
In Alarm::CallBack(),totalTicks=4900
In Alarm::CallBack(),totalTicks=5300
In Alarm::CallBack(),totalTicks=5700
In Alarm::CallBack(),totalTicks=6100
In Alarm::CallBack(),totalTicks=6500
In Alarm::CallBack(),totalTicks=6900
In Alarm::CallBack(),totalTicks=7300
In Alarm::CallBack(),totalTicks=7700
In Alarm::CallBack(),totalTicks=8100
In Alarm::CallBack(),totalTicks=8500
In Alarm::CallBack(),totalTicks=8900
In Alarm::CallBack(),totalTicks=9300
In Alarm::CallBack(),totalTicks=9700
In Alarm::CallBack(),totalTicks=10100
In Alarm::CallBack(),totalTicks=10500
```

Question 7

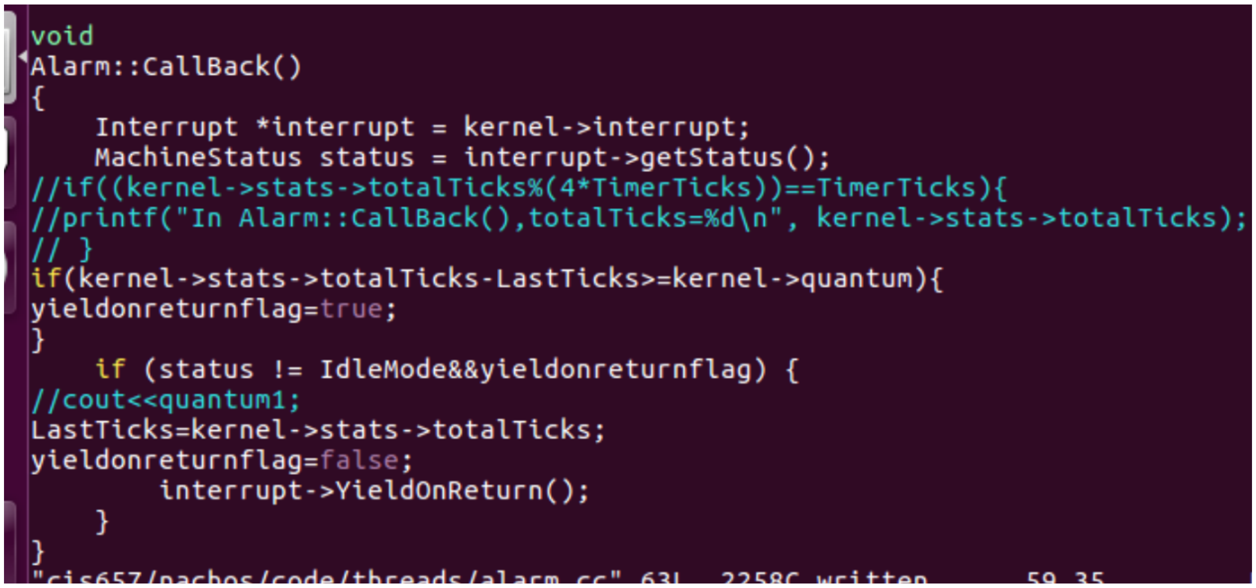
Firstly, we follow the “-rs” to write the “-quantum” flag. Implementing the argument parsing for the “quantum” flag is the following code in kernel.cc.



```
for (int i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-rs") == 0) {
        ASSERT(i + 1 < argc);
        RandomInit(atoi(argv[i + 1]));// initialize pseudo-rando
                                      // number generator
        randomSlice = TRUE;
        i++;
    } else if (strcmp(argv[i], "-quantum")==0){
        ASSERT(i + 1 < argc);
        quantum=atoi(argv[i+1]);
        i++;

      else if (strcmp(argv[i], "-s") == 0) {
        debugUserProg = TRUE;
    } else if (strcmp(argv[i], "-ci") == 0) {
        ASSERT(i + 1 < argc);
        consoleIn = argv[i + 1];
        i++;
    } else if (strcmp(argv[i], "-co") == 0) {
        ASSERT(i + 1 < argc);
        consoleOut = argv[i + 1];
        i++;
```

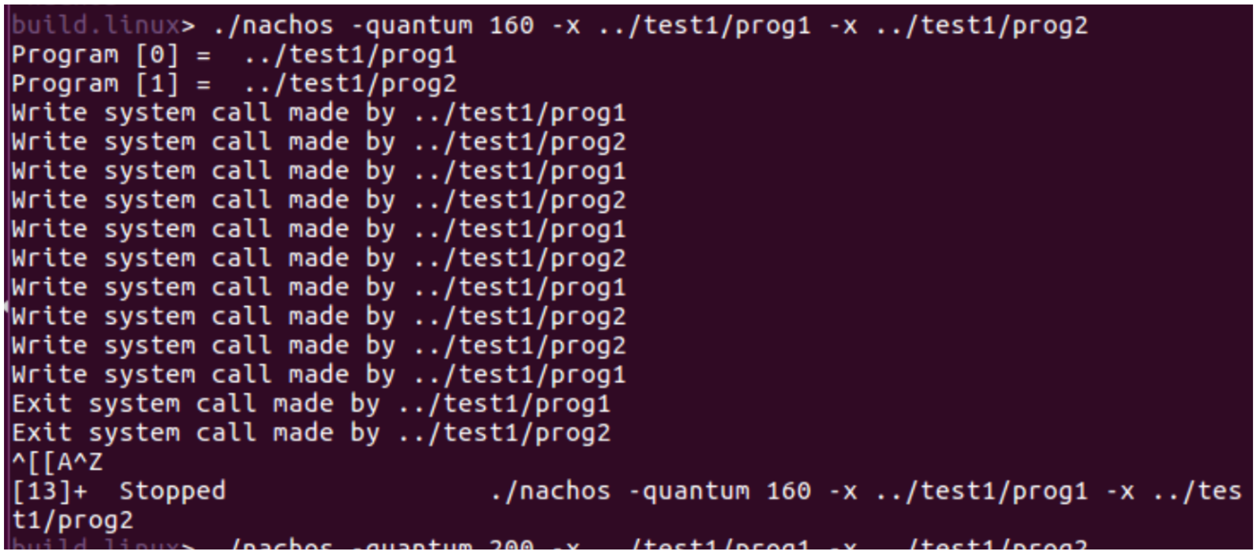
In order to change the setting interrupt->YieldOnReturn flag, we need another flag that is only set after the given “quantum”, which is yieldonreturnflag in my code(default is false). After getting the given “quantum” it will be set true to let the interrupt->YieldOnReturn happen. That’s the modified alarm.cc



```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
//if((kernel->stats->totalTicks%(4*TimerTicks))==TimerTicks){
//printf("In Alarm::CallBack(),totalTicks=%d\n", kernel->stats->totalTicks);
// }
if(kernel->stats->totalTicks-LastTicks>=kernel->quantum){
yieldonreturnflag=true;
}
    if (status != IdleMode&&yieldonreturnflag) {
//cout<<quantum1;
LastTicks=kernel->stats->totalTicks;
yieldonreturnflag=false;
        interrupt->YieldOnReturn();
    }
}
"cis657/nachos/code/threads/alarm.cc" 63L, 2258C written        59,35
```

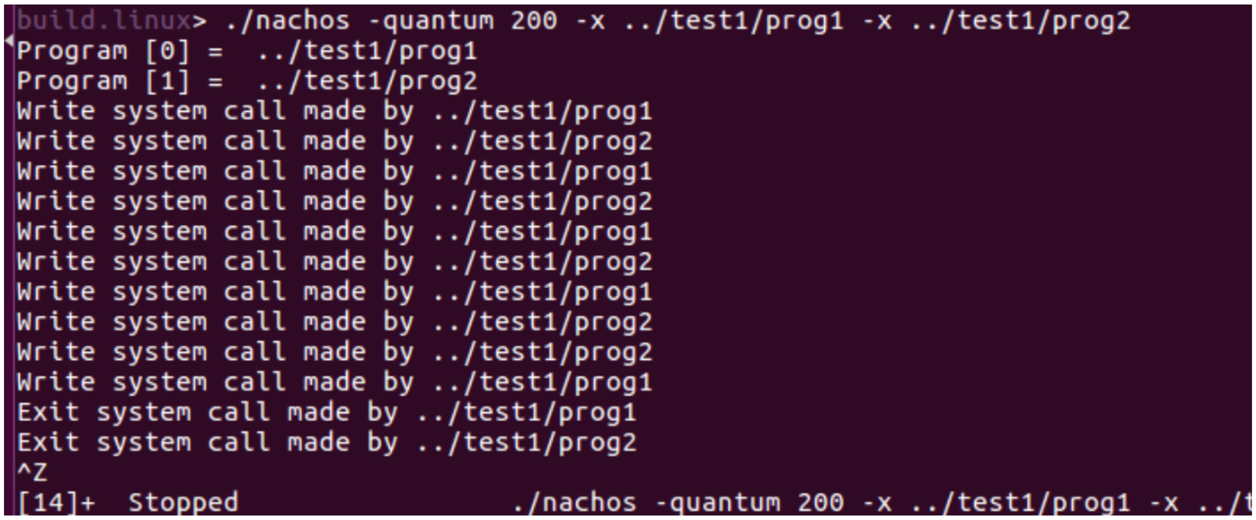
Due to the requirement of the 7.2. The actual quantum is rounded to the next multiple of 100 for any value. In order to prove, the 160 and 200 of the quantum should be the same, which is as

follows. The output is because that we will only set the yieldonreturnflag true after the given number of ticks. So before the totalTicks-LastTicks< quantum. The running program won't yieldonreturn the CPU. Only totalTIcks-LastTicks>quantum. The running prog1 will give up the CPU, and let prog2 to run. Because the quantum=160 or 200, so the prog will firstly interrupt->YiledOnReturn at the totalTicks=200. And then prog2 start using CPU and give up CPU when the totalTicks=400. And keeping doing this will output the following result.
(Not a multiple of 100)



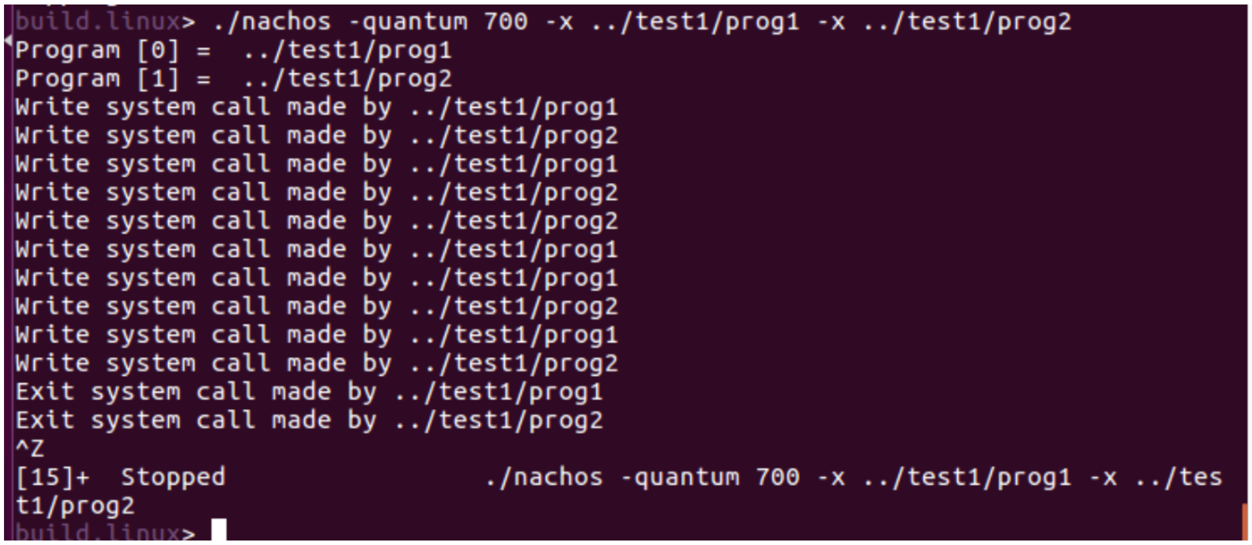
```
build.linux> ./nachos -quantum 160 -x ../test1/prog1 -x ../test1/prog2
Program [0] =  ../test1/prog1
Program [1] =  ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Exit system call made by ../test1/prog1
Exit system call made by ../test1/prog2
^[[A^Z
[13]+  Stopped                 ./nachos -quantum 160 -x ../test1/prog1 -x ../tes
t1/prog2
build.linux> ./nachos -quantum 200 -x ../test1/prog1 -x ../test1/prog2
```

(a multiple of 100)



```
build.linux> ./nachos -quantum 200 -x ../test1/prog1 -x ../test1/prog2
Program [0] =  ../test1/prog1
Program [1] =  ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Exit system call made by ../test1/prog1
Exit system call made by ../test1/prog2
^Z
[14]+  Stopped                 ./nachos -quantum 200 -x ../test1/prog1 -x ../t
```

When the quantum is 700. Prog1 will interrupt->YieldOnReturn when the totalTicks equals to 700. Then the prog2 starts to run and gives up when the totalTicks equals to 1400. So keep doing this.



```
build.linux> ./nachos -quantum 700 -x ../test1/prog1 -x ../test1/prog2
Program [0] =  ../test1/prog1
Program [1] =  ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Write system call made by ../test1/prog1
Write system call made by ../test1/prog2
Exit system call made by ../test1/prog1
Exit system call made by ../test1/prog2
^Z
[15]+  Stopped                 ./nachos -quantum 700 -x ../test1/prog1 -x ../tes
t1/prog2
build.linux> 
```