CIS657 Spring 2019

Assignment 1

Submitted by –

Name – Anish Nesarkar

SUID – 368122582

Subject – Operating Systems

Assignment 1

CIS657 Spring 2019

Assignment Disclosure Form

Assignment #: 1

Name: Anish Nesarkar

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

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?

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

Signature:____________Anish Nesarkar________________ Date : 2/29/2019

<u>Design and Implementation:</u>

A multiple level feedback queue scheduling is being implemented using a simulated I/O. Four levels of ready queues with different amount of time quantum are used. Four flags are used q1, q2, q3 and q4 for the quantum values at each level.

There threads are created which are CPU-bound, IO-bound and mixed. The CPU-bound threads do the computing operations in the CPU. The mixed threads consist of computing operations and IO operations. The IO-bound thread consists of the following functions:

I/O simulating input(read)/output(write) operations are designed as follows:

A request object is created to store the information and state of the thread using I/O operations.

Write Operation:

- A new request object is created with this routine is called.
- A random delay is set for the operation and the completion is calculated using current time and the information is stored in the request object.
- The following information is stored in the request object
    - Output string to be written
    - Current thread pointer
    - Random delay between 200 and 300 units.
    - Type of operation. i.e “Write”
    - Completion time
    - The priority is stored in the thread
- The request is then added to the event queue
- An IO interrupt is registered in the IOAlarm
- The IO thread is put to sleep

Read Operation:

- A new request object is created with this routine is called.
- A random delay is set for the operation and the completion is calculated using current time and the information is stored in the request object.
- The following information is stored in the request object
    - Char pointer to be read
    - Current thread pointer
    - Random delay between 400 and 500 units.
    - Type of operation. i.e “Read”
    - Completion time of the IO operation
    - The priority is stored in the thread

- The request is then added to the event queue
- An IO interrupt is registered in the IOAlarm
- The IO thread is put to sleep

EventQueue – This queue consists of request objects which are arranged in ascending order of the completion time of the IO request.

ThreadTest

```
//creating instance of IOInterrupt to call IOWrite and IORead
IOInterrupt *IOInt;
string s;

//Function to carry out IO Operations
void IOFunction(int i)
{
    cout << endl;
    cout << ">----------------------< IO-bound function >----------------------
<"<< endl;
    cout << "This is IO function for Thread t"<< kernel->currentThread-
>getThreadID() << endl;
    //IOWrite operation that calls IOInterrupt
    IOInt->IOWrite("Operating System ");
    //IORead operation that calls Interrupt
    string text = IOInt->IORead(s);
    IOInt->IOWrite("Assignment 1");
    cout << endl;
    cout << ">-------------------------------------------------------------------
--------<" << endl;
    cout << "Finishing thread IO-bound thread with thread ID: " << kernel-
>currentThread->getThreadID() << endl;
    cout << ">-------------------------------------------------------------------
--------<" << endl << endl;
    kernel->currentThread->Finish();
}

//Function to carry out CPU computing operation
void ComputeFunction(int i)
{
    cout << endl;
    cout << ">----------------------< CPU-bound function >----------------------
<"<< endl;
    cout << "This is compute function for thread t"<< kernel->currentThread-
>getThreadID() << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;
    kernel->interrupt->OneTick();
    cout << "Multiplication : 134 * 243 = " << 134 * 243 << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;

    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    cout << endl;
    cout << endl;
    cout << ">-------------------------------------------------------------------
--------<" << endl;
    cout << "Finishing thread CPU-bound thread with thread ID: " << kernel-
>currentThread->getThreadID() << endl;
    cout << ">-------------------------------------------------------------------
--------<" << endl << endl;
```

```
    kernel->currentThread->Finish();
}

//function to carry out combination of computing operation and IO operation
void Hybrid(int i)
{
    cout << ">----------------------< Hybrid function >----------------------<"<<
endl;
    cout << "This is Hybrid function pointer for thread t" << kernel-
>currentThread->getThreadID() << endl;
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    //IOWrite operation that calls IOInterrupt
    IOInt->IOWrite("Hybrid function ");
    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }

    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }

    for(int j = 1; j < 21 ; j++)
    {
        kernel->interrupt->OneTick();
        cout << j << " ";
    }
    //IORead operation that calls IOInterrupt for Read
    string text1 = IOInt->IORead(s);
    cout << endl;
    cout << ">-----------------------------------------------------------------
--------<" << endl;
    cout << "Finishing Mixed thread with thread ID: " << kernel->currentThread-
>getThreadID() << endl;
    cout << ">-----------------------------------------------------------------
--------<" << endl << endl;
    //Finishing the Hybrid Thread
    kernel->currentThread->Finish();
```

```
}

//------------< Defined function that creates child threads which are CPU-bound,
IO bound or Hybrid for executing program >--------------
void myFunction()
{
cout << endl;
cout << "-----------------------< Simulating MLFQ in Nachos >--------------------
-----" << endl;
cout << endl;
cout << "Number of levels is MLFQ : 4" << endl;
cout << "Level 1 has priority 4 with Quantum q1 = " << kernel->q1 << endl;
cout << "Level 2 has priority 3 with Quantum q2 = " << kernel->q2 << endl;
cout << "Level 3 has priority 2 with Quantum q3 = " << kernel->q3 << endl;
cout << "Level 4 has priority 1 with Quantum q4 = " << kernel->q4 << endl;
cout << endl;
cout << "The number threads forked for the simulation : 10" << endl << endl;

cout << "Number of IO-bound threads : 4" << endl;
cout << "Number of CPU-bound threads : 4" << endl;
cout << "Number of Mixed threads : 2" << endl;

//Enabling the interrupts
kernel->interrupt->Enable();

//Forking 5 IO-bound threads

Thread *t1 = new Thread("IO bound thread t1");
t1->setThreadID(1);
t1->setThreadPriority(4);
t1 -> Fork((VoidFunctionPtr) IOFunction, (void *) 1);

Thread *t2 = new Thread("IO bound thread t2");
t2->setThreadID(2);
t2->setThreadPriority(4);
t2 -> Fork((VoidFunctionPtr) IOFunction, (void *) 2);

Thread *t3 = new Thread("IO bound thread t3");
t3->setThreadID(3);
t3->setThreadPriority(4);
t3->Fork((VoidFunctionPtr) IOFunction, (void *) 3);

Thread *t4 = new Thread("IO bound thread t4");
t4->setThreadID(4);
t4->setThreadPriority(4);
```

```
t4->Fork((VoidFunctionPtr) IOFunction, (void *) 4);

Thread *t5 = new Thread("IO bound thread t5");
t5->setThreadID(5);
t5->setThreadPriority(4);
t5->Fork((VoidFunctionPtr) IOFunction, (void *) 5);

//forking 5 CPU bound threads

Thread *t6 = new Thread("CPU bound thread t6");
t6->setThreadID(6);
t6->setThreadPriority(4);
t6->Fork((VoidFunctionPtr) ComputeFunction, (void *) 6);


Thread *t7 = new Thread("CPU bound thread t7");
t7->setThreadID(7);
t7->setThreadPriority(4);
t7 -> Fork((VoidFunctionPtr) ComputeFunction, (void *) 7);


Thread *t8 = new Thread("CPU bound thread t8");
t8->setThreadID(8);
t8->setThreadPriority(4);
t8 -> Fork((VoidFunctionPtr) ComputeFunction, (void *) 8);

Thread *t9 = new Thread("CPU bound thread t9");
t9->setThreadID(9);
t9->setThreadPriority(4);
t9 -> Fork((VoidFunctionPtr) ComputeFunction, (void *) 9);

Thread *t10 = new Thread("CPU bound thread t10");
t10->setThreadID(10);
t10->setThreadPriority(4);
t10->Fork((VoidFunctionPtr) ComputeFunction, (void *) 10);

//forking 2 Hybrid threads

Thread *t11 = new Thread("Hybrid thread t11");
t11->setThreadID(11);
t11->setThreadPriority(4);
t11->Fork((VoidFunctionPtr) Hybrid, (void *) 11);

Thread *t12 = new Thread("Hybrid thread t12");
t12->setThreadID(12);
```

```
t12->setThreadPriority(4);
t12-> Fork((VoidFunctionPtr) Hybrid, (void *) 12);

//finishing the main thread
kernel->currentThread->Finish();

}
```

In threadtest, the flow of the code is follows:

- The myFunction() forks the threads which are either compute-bound, IO-bound or mixed.
- The IO-bound function pointer has IO operations which are either IOWrite or IORead.
- The compute thread has computing operations like for loops.
- The Hybrid thread has a mix of compute operations and IO operations.
- All these threads are finished at the end of the functions.

IOInterrupt

```
//constructor for IOInterrupt
IOInterrupt::IOInterrupt()
{

}

//IO Write interrupt
void IOInterrupt::IOWrite(string s)
{
    //create new request to store following details
    request *r = new request();
    r->setWriteOutput(s);
    r->setThreadState(kernel->currentThread);
    r->setDelay(200 + (rand() % (101)));
    r->setType("Write");
    //set the completion time of the IO operation
    r->setCompletionTime(kernel->stats->totalTicks + r->getDelay());
    cout << endl << endl;
    cout << ">--------------------------< Register Write >-----------------------
<" << endl;
    cout << "Register Write with Thread ID: " << kernel->currentThread-
>getThreadID() << endl;
    cout << "The Write request will be served after the time : " << r-
>getCompletionTime() << endl;
    cout << "The priority of the thread is initially set to : " << kernel-
>currentThread->getThreadPriority() << endl;
```

```
    cout << ">------------------------------------------------------------------
<" << endl << endl;
    //insert the request into the event queue
    kernel->eventQueue->Insert(r);
    //set the interrupt for the random delay
    kernel->IOalarm->IOtimer->SetInterrupt(r->getDelay());
    //put the thread to sleep
    kernel->interrupt->SetLevel(IntOff);
    kernel->currentThread->Sleep(FALSE);
}

//IO interrupt for Read
string IOInterrupt::IORead(string s)
{
    //create new request to store following details
    request *r = new request();
    s = "A Read string";
    r->setReadOutput(s);
    r->setThreadState(kernel->currentThread);
    r->setDelay(400 + (rand() % (101)));
    r->setType("Read");
    //set the completion time of the IO operation
    r->setCompletionTime(kernel->stats->totalTicks + r->getDelay());
    cout << endl << endl;
    cout << ">--------------------------< Register Read >-----------------------
<" << endl;
    cout << "Register Read with Thread ID : " << kernel->currentThread-
>getThreadID() << endl;
    cout << "The Read request will be served after the time : " << r-
>getCompletionTime() << endl << endl;
    cout << ">------------------------------------------------------------------
<" << endl << endl;
    //insert the request into the event queue
    kernel->eventQueue->Insert(r);
    //set the interrupt for the random delay
    kernel->IOalarm->IOtimer->SetInterrupt(r->getDelay());
     //put the thread to sleep
    kernel->interrupt->SetLevel(IntOff);
    kernel->currentThread->Sleep(FALSE);
    return s;
}
```
In IOInterrupt, the flow of code is as follows:

- This consists of IOWrite and IORead interrupts
- A request object pointer is created

- The output to be written or read is stored in the request object.
- The threadstate, delay,type and completion time are also stored in the request object.
- The request object is put into the event queue.
- The Interrupt is registered into the IOAlarm
- The thread is then put to sleep and woken up when its time to serve the request has arrived.

IOTimer

```
//IOTimer for the IO operations
IOTimer::IOTimer(bool doRandom, CallBackObj *toCall)
{
    randomize = doRandom;
    callPeriodically = toCall;
    disable = FALSE;
}

//IO Timer callback for the IO operations
void
IOTimer::CallBack()
{
    callPeriodically->CallBack();
}

//Set interrupts for the IO operations
void
IOTimer::SetInterrupt(int d)
{
       kernel->interrupt->Schedule(this, d, IOTimerInt);
}
```

- The IOTimer is used for setting the interrupts for the IO operations.
- The callback function of IOTimer is called when the Interrupt occurs.

IOAlarm

```
//constructor for IO Alarm
IOAlarm::IOAlarm(bool doRandom)
```

```
{
  //create object of IOtimer for IO operations
    IOtimer = new IOTimer(doRandom, this);
}


//IOAlarm callback when the interrupt occurs
void
IOAlarm::CallBack()
{
    //list iterator for the event queue
    ListIterator<request*> *eQueuelist = new ListIterator<request*>(kernel-
>eventQueue);
    //IOHandler instance
    IOHandler *IHandler;
    while(!eQueuelist->IsDone())
    {
        request *r = eQueuelist->Item();
        //check if handler is write
        if(r->getType() == "Write" && (kernel->stats->totalTicks >= r-
>getCompletionTime()))
        {
            IHandler->write(r);
        }
        //check if handler is read
        else if(r->getType() == "Read" && (kernel->stats->totalTicks >= r-
>getCompletionTime()))
        {
            IHandler->read(r);
        }
        eQueuelist->Next();
    }
}
```

- The callback function of IOAlarm is called from the callback of the timer.
- The EventQueue is iterated for all the requests.
- If the current time is greater than completion time of the IO request, the IO request is served.

IOHandler

```
//constructor for IO Handler
IOHandler::IOHandler()
```

```
{
}

//Write IO handler
void IOHandler::write(request *r)
{
    //remove the served IO from the event queue
    kernel->eventQueue->Remove(r);
    Thread *t = r->getThreadState();
    int priority = t->getThreadPriority();
    cout << endl << endl;
    cout << ">------------< Serving IO Write request >--------------<" << endl;
    cout << "Write IOHandler for thread t" << t->getThreadID() << " with priority
: " <<  priority << endl;
    cout << "Wrote : " << r->getWriteOutput() << endl;
    //increment the priority of the thread by 1. Do not increment if already in
highest priority
    if(priority < 4)
    {
        priority++;
        t->setThreadPriority(priority);
    }
    cout << "Dynamically changing Priority of the thread t" << t->getThreadID()<<
" from "<< (priority - 1) << " to " << priority << endl;

    cout << "Write request served at time : " << kernel->stats->totalTicks <<
endl;
    cout << ">---------------------------------------------------------<"<< endl
<< endl;
    //put the thread to ready to run
    kernel->scheduler->ReadyToRun(t);
}

void IOHandler::read(request *r)
{
    //remove the served IO from the event queue
    kernel->eventQueue->Remove(r);
    cout << endl << endl;
    cout << ">------------< Serving IO Read request >--------------< " << endl;
    Thread *t = r->getThreadState();
    int priority = t->getThreadPriority();
    cout << "Read IOHandler for thread t" << t->getThreadID() << " with priority
: " <<  priority << endl;
    cout << "Information Read : " << r->getReadOutput() << endl;
    //increment the priority of the thread by 1. Do not increment if already in
highest priority
    if(priority < 4)
    {
        priority++;
        t->setThreadPriority(priority);
    }
    cout << "Dynamically changing Priority of the thread t" << t->getThreadID()<<
" from "<< (priority - 1) << " to " << priority << endl;

    cout << "Read request served at time : " << kernel->stats->totalTicks <<
endl;
    cout << ">----------------------------------------------------------<"<< endl
<< endl;
    //put the thread to ready to run
    kernel->scheduler->ReadyToRun(t);

}
```

- In IOhandler, the IO interrupts are served.
- The priority of the IO-bound or mixed thread is incremented by 1.
- The thread is put to ready to run state.

Alarm Callback and Timer SetInterrupt

```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    if (status != IdleMode) {
    interrupt->YieldOnReturn();
    }
}
```

```
void
Timer::SetInterrupt(int quantum)
{
       // schedule the next timer device interrupt
       kernel->interrupt->Schedule(this, quantum, TimerInt);
}
```

In Alarm,

- A timer object is created in Alarm constructor to register the interrupt
- The Callback function is called for every interrupt registered in Timer.cc
- The current thread is yielded and next thread is run in the callback of the alarm.
- The setInterrupt of the Timer takes parameters as quantums and registers the timer for those quantums (q1,q2,q3,q4).

Scheduler

```
Scheduler::ReadyToRun (Thread *thread)
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    DEBUG(dbgThread, "Putting thread on ready list: " << thread->getName());

    thread->setStatus(READY);

//Putting the threads to respective levels based on their priority and storing the time when the list was put into the ready list

    if(thread->getThreadPriority() >= 4)
        Level1->Append(thread);
    else if(thread->getThreadPriority() == 3)
    {
        thread->setReadyListTime(kernel->stats->totalTicks);
        Level2->Append(thread);
    }
    else if(thread->getThreadPriority() == 2)
    {
        thread->setReadyListTime(kernel->stats->totalTicks);
        Level3->Append(thread);
    }
    else if(thread->getThreadPriority() == 1)
    {
        thread->setReadyListTime(kernel->stats->totalTicks);
        Level4->Append(thread);
    }
}
```

- Four lists are created for respective levels Level1, Level2, Level3 and Level4.
- The threads are appended to the respective levels based on the priorities.
    - A thread with priority 4 will be added to Level 4
    - A thread with priority 3 will be added to level 3
    - A thread with priority 2 will be added to level 2

    - A thread with priority 1 will be added to level 1
- The time is set when the thread is added to ready list which is then used for aging.

FindNextToRun

```
Thread *
Scheduler::FindNextToRun ()
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    Alarm *alarmobj = kernel->alarm;
    Timer *timerobj = alarmobj->timer;

 //Performing aging to prevent starvation
    int age;
    //check if level is empty
     if(!Level2->IsEmpty())
     {
        //iterate through the list
        ListIterator<Thread*> *Level2list = new ListIterator<Thread*>(Level2);
        while(!Level2list->IsDone())
        {
            Thread *t = Level2list->Item();
            //calculate the age of the corresponding thread
            age = kernel->stats->totalTicks - t->getReadyListTime();
            //if the age is greater than the threshold i.e. 400, put the thread 
to higher order ready queue
            if(age > 400)
            {
                cout << endl << endl;
                cout << ">-----------------------------------< Aging >----------
-----------------------<" << endl;
                cout << "Moving the thread " << t->getName() << " from Level 2 
to Level 1 because of Starvation " << endl;
               cout << "Dynamically changing priority "<< t->getName() << " 
from 3 to 4" << endl;

                cout << ">-----------------------------------------------------
-----------------------<" << endl << endl;
                //set the priority of the thread
                t->setThreadPriority(4);
                Level1->Append(t);
                Level2->Remove(t);
                ListIterator<Thread*> *Level2list = new 
ListIterator<Thread*>(Level2);
```

```
            }
            Level2list->Next();
        }
    }

    //check if level is empty
    if(!Level3->IsEmpty())
    {
        //iterate through the list
        ListIterator<Thread*> *Level3list = new ListIterator<Thread*>(Level3);
        while(!Level3list->IsDone())
        {
            Thread *t = Level3list->Item();
            //calculate the age of the corresponding thread
            age = kernel->stats->totalTicks - t->getReadyListTime();
            //if the age is greater than the threshold i.e. 400, put the thread
to higher order ready queue
            if(age > 400)
            {
                cout << endl << endl;
                cout << ">-------------------------------< Aging >--------------
-----------------<" << endl;
                cout << "Moving the thread " << t->getName() << " from Level 3
to Level 2 because of Starvation " << endl;
                cout << "Dynamically changing priority "<< t->getName() << "
from 2 to 3" << endl;

                cout << ">------------------------------------------------------
-----------------<" << endl << endl;
                //set the priority of the thread
                t->setThreadPriority(3);
                Level2->Append(t);
                Level3->Remove(t);
            }
            Level3list->Next();
        }
    }

    //check if level is empty
    if(!Level4->IsEmpty())
    {
        //iterate through the list
        ListIterator<Thread*> *Level4list = new ListIterator<Thread*>(Level4);
        while(!Level4list->IsDone())
        {
             Thread *t = Level4list->Item();
             //calculate the age of the corresponding thread
             age = kernel->stats->totalTicks - t->getReadyListTime();
             //if the age is greater than the threshold i.e. 400, put the thread
to higher order ready queue
             if(age > 300)
             {
                 cout << endl << endl;
                 cout << ">------------------------------------< Aging >--------
---------------------------<" << endl;
                 cout << "Moving the thread " << t->getName() << " from Level 4
to Level 3 because of Starvation " << endl;
                 cout << "Dynamically changing priority "<< t->getName() << "
from 1 to 2" << endl;


                 cout << ">-----------------------------------------------------
---------------------------<" << endl << endl;
                 //set the priority of the thread
                 t->setThreadPriority(3);
                 Level3->Append(t);
                 Level4->Remove(t);
             }
             Level4list->Next();
         }
     }

    //Removing the next thread to be run and removing pending interrupts if any
for sleeping or finished threads
    if(!Level1->IsEmpty())
    {
        //iterate through pending interrupts list
               ListIterator<PendingInterrupt *> *pendinglist = new
ListIterator<PendingInterrupt *>(kernel->interrupt->pending);
               while(!pendinglist->IsDone())
               {
                   //remove TimerInt thread
                   if(pendinglist->Item()->type == 0)
                   {
                       kernel->interrupt->pending->Remove(pendinglist->Item());
                       break;
                   }
                   pendinglist->Next();
               }
               //set the interrupt for quantum value for level 1
```

```
                timerobj->SetInterrupt(kernel->q1);
                return Level1->RemoveFront();
    }
    else if(!Level2->IsEmpty())
    {
        //iterate through pending interrupts list
        ListIterator<PendingInterrupt *> *pendinglist = new
ListIterator<PendingInterrupt *>(kernel->interrupt->pending);
               while(!pendinglist->IsDone())
               {
                   //remove TimerInt thread
                   if(pendinglist->Item()->type == 0)
                   {
                       kernel->interrupt->pending->Remove(pendinglist->Item());
                       break;
                   }
                   pendinglist->Next();
               }
               //set the interrupt for quantum value for level 2
                timerobj->SetInterrupt(kernel->q2);
                return Level2->RemoveFront();
    }
    else if(!Level3->IsEmpty())
    {
        //iterate through pending interrupts list
        ListIterator<PendingInterrupt *> *pendinglist = new
ListIterator<PendingInterrupt *>(kernel->interrupt->pending);
               while(!pendinglist->IsDone())
               {
                    //remove TimerInt thread
                   if(pendinglist->Item()->type == 0)
                   {
                       kernel->interrupt->pending->Remove(pendinglist->Item());
                       break;
                   }
                   pendinglist->Next();
               }
               //set the interrupt for quantum value for level 3
                timerobj->SetInterrupt(kernel->q3);
                return Level3->RemoveFront();
    }
    else if(!Level4->IsEmpty())
    {
        //iterate through pending interrupts list
        ListIterator<PendingInterrupt *> *pendinglist = new
ListIterator<PendingInterrupt *>(kernel->interrupt->pending);
               while(!pendinglist->IsDone())
               {
                   //remove TimerInt thread
                   if(pendinglist->Item()->type == 0)
                   {
                       kernel->interrupt->pending->Remove(pendinglist->Item());
                       break;
                   }
                   pendinglist->Next();
               }
                //set the interrupt for quantum value for level 4
                timerobj->SetInterrupt(kernel->q4);
                return Level4->RemoveFront();
    }
    else
        return NULL;

}
```

- Aging is performed in this function.
- The thread is moved up the levels and the priorities are changed if the time the thread is present in that ready list is greater than a threshold value. i.e. 400.
- Next thread is removed for the levels according to the level order. i.e. leve1, level2, level3, level4.
- The pending interrupts are removed if any that are set by the quantums.
- Four quantums are set for four levels before removing the thread to run next.
- The default values of quantums are
    - q1 – 200
    - q2 – 400
    - q3 – 600
    - q4 - 800

```
//changing the priority of the thread after the context switch
 int priority = kernel->currentThread->getThreadPriority();
     switch(priority)
     {
         case 4 : {
             //setting thread priority to 3
             kernel->currentThread->setThreadPriority(3);
             cout << endl << endl;
```

```
            cout << ">-----------------------< Context Switch >-----------------
<" << endl;
            cout << "Switching from " << kernel->currentThread->getName() << "
to " << nextThread->getName()<< " : Time quantum q1 = "<< kernel->q1 << "
expired" <<endl;

            cout << "setting priority of thread " << kernel->currentThread-
>getName() << " to 3" << endl;
            cout << ">----------------------------------------------------------
-<" << endl << endl;
            break;
        }
        case 3 : {
            //setting thread priority to 2
            kernel->currentThread->setThreadPriority(2);
            cout << endl << endl;
            cout << ">-----------------------< Context Switch >-----------------
<" << endl;
            cout << "Switching from " << kernel->currentThread->getName() << "
to " << nextThread->getName() << " : Time quantum q2 = "<< kernel->q2 << "
expired" << endl;

            cout << "setting priority of thread " << kernel->currentThread-
>getName() << " to 2" << endl;
            cout << ">----------------------------------------------------------
-<" << endl << endl;
            break;
        }
        case 2 : {
            //setting thread priority to 1
            kernel->currentThread->setThreadPriority(1);
            cout << endl << endl;
            cout << ">-----------------------< Context Switch >-----------------
<" << endl;
            cout << "Switching from " << kernel->currentThread->getName() << "
to " << nextThread->getName() << " : Time quantum q3 = "<< kernel->q3 << "
expired" << endl;

            cout << "setting priority of thread " << kernel->currentThread-
>getName() << " to 1" << endl;
            cout << ">----------------------------------------------------------
-<" << endl << endl;
            break;
        }
        case 1 : {
            //keeping the thread priority 1 for lowest level
```

```
             kernel->currentThread->setThreadPriority(1);
             cout << endl << endl;
             cout << ">-----------------------< Context Switch >-----------------
<" << endl;
             cout << "Switching from " << kernel->currentThread->getName() << "
to " << nextThread->getName() << " : Time quantum q4 = "<< kernel->q4 << "
expired" << endl;

             cout << "Keeping priority of thread " << kernel->currentThread-
>getName() << " to 1 because it is lowest level" << endl;
             cout << ">---------------------------------------------------------
-<" << endl << endl;
             break;
         }
      }
```

- The priority of the thread is decremented by 1 when it has context switched.
- If the thread has priority 1, the priority of that thread remains the same.

Testing

How to run the Test

1. Copy the nachos folder from your local machine to the server.
2. Copy the header files request.h, IOAlarm.h, IOInterrupt.h, IOTimer.h, IOHandler.h and class files request.cc, IOAlarm.cc, IOInterrupt.cc, IOTimer.cc, IOHandler.cc to the location /nachos/code/threads/
3. Copy the modified Threadtest.cc from the local machine to the server to the location /nachos/code/threads/
4. Navigate to the directory nachos/code/build.linux
5. Execute the below commands
    a. make clean
    b. make depend
    c. make nachos
    d. ./nachos -K

Added Files:

request.cc, IOAlarm.cc, IOInterrupt.cc, IOTimer.cc, IOHandler.cc and request.h, IOAlarm.h, IOInterrupt.h, IOTimer.h, IOHandler.h to the location /nachos/code/threads/

Modified Files:

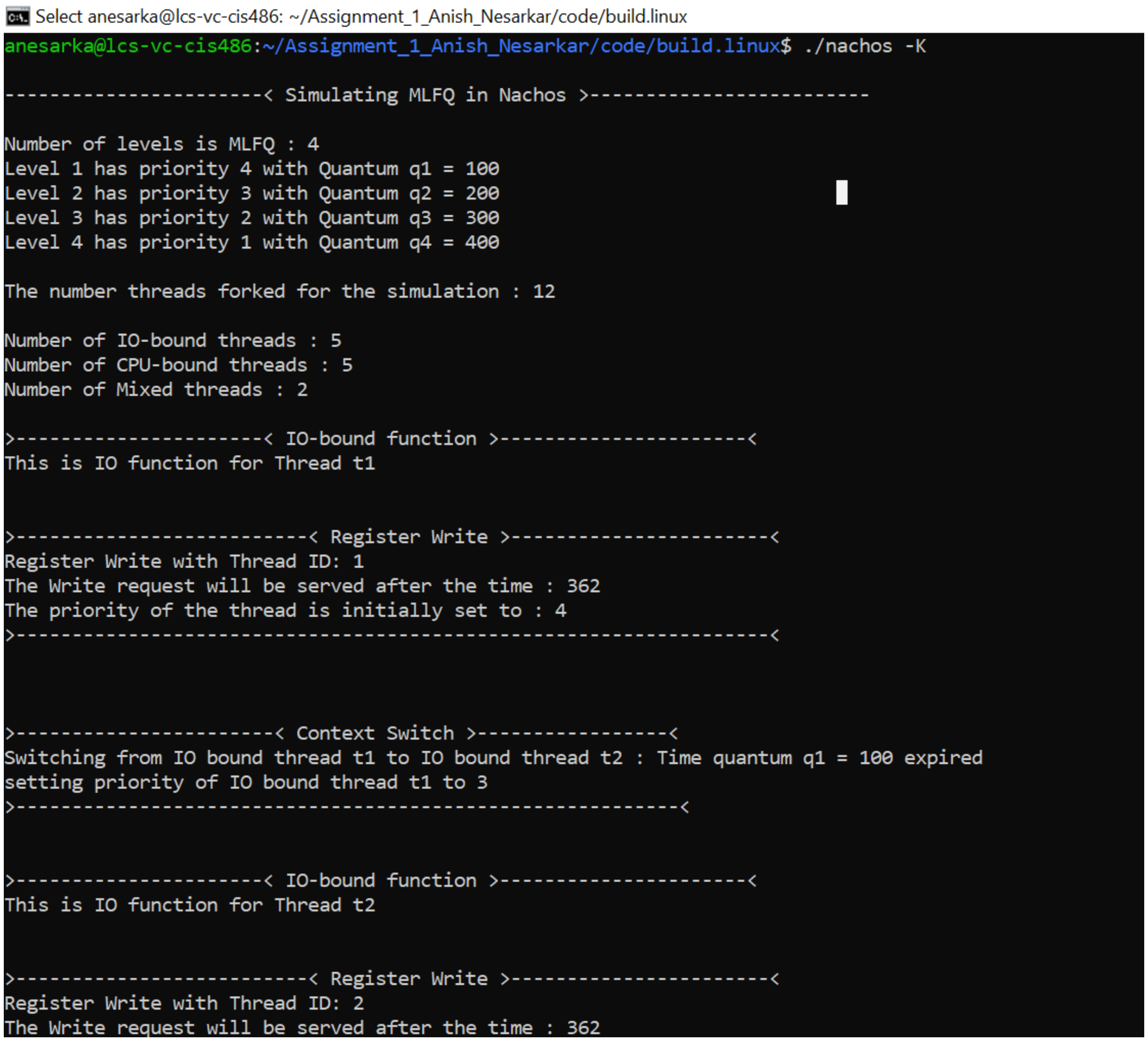
Threadtest.cc in the location /nachos/code.threads/

Makefile

```
THREAD_H = ../threads/alarm.h\
    ../threads/kernel.h\
    ../threads/main.h\
    ../threads/scheduler.h\
    ../threads/switch.h\
    ../threads/synch.h\
    ../threads/synchlist.h\
    ../threads/thread.h\
    ../threads/request.h\
    ../threads/IOInterrupt.h\
    ../threads/IOHandler.h\
    ../threads/IOAlarm.h\
    ../threads/IOTimer.h

THREAD_C = ../threads/alarm.cc\
    ../threads/kernel.cc\
    ../threads/main.cc\
    ../threads/scheduler.cc\
    ../threads/synch.cc\
    ../threads/synchlist.cc\
    ../threads/thread.cc\
    ../threads/threadtest.cc\
    ../threads/request.cc\
    ../threads/IOInterrupt.cc\
    ../threads/IOHandler.cc\
    ../threads/IOAlarm.cc\
    ../threads/IOTimer.cc

THREAD_O = alarm.o kernel.o main.o scheduler.o synch.o thread.o threadtest.o
request.o IOInterrupt.o IOHandler.o IOAlarm.o IOTimer.o
```

OUTPUT

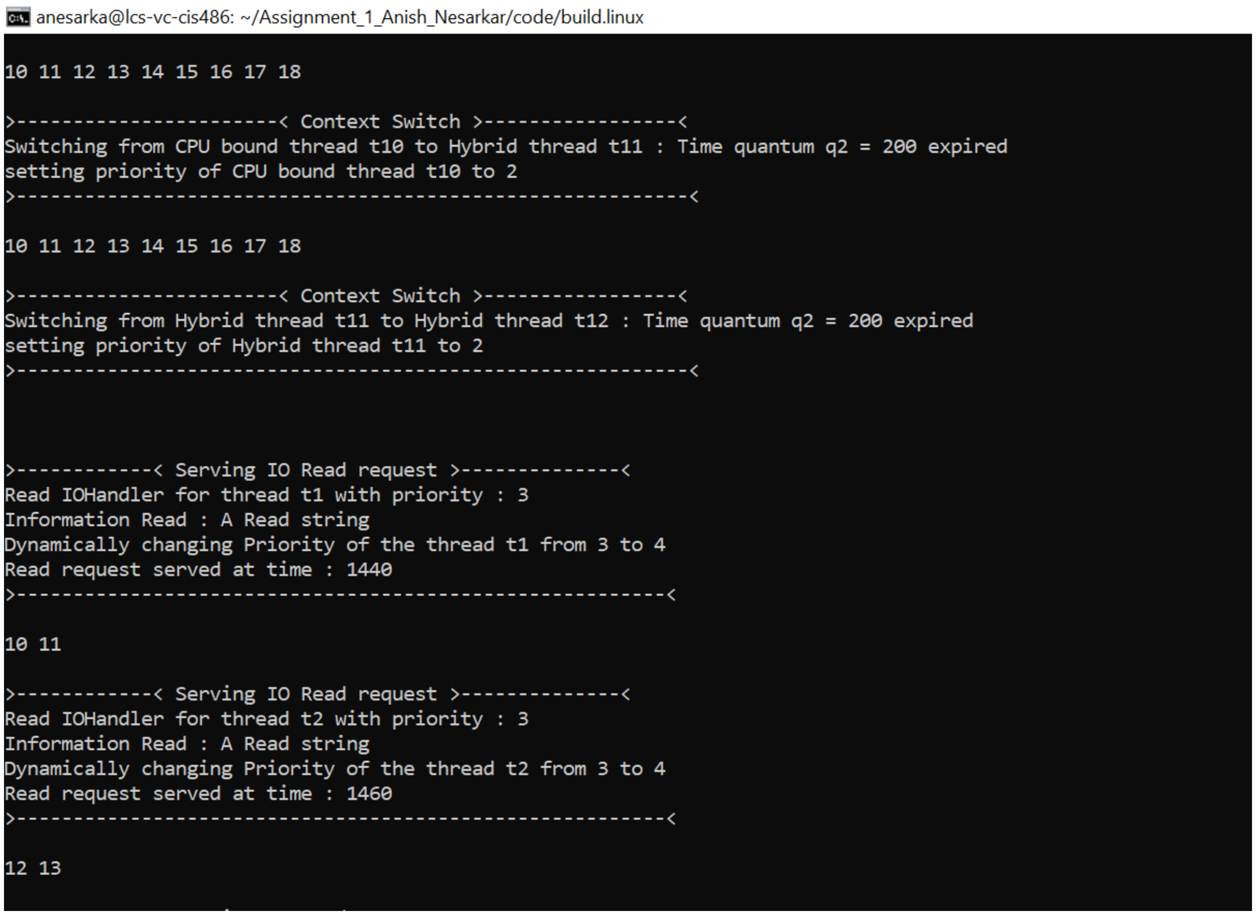
1. Thread spawning



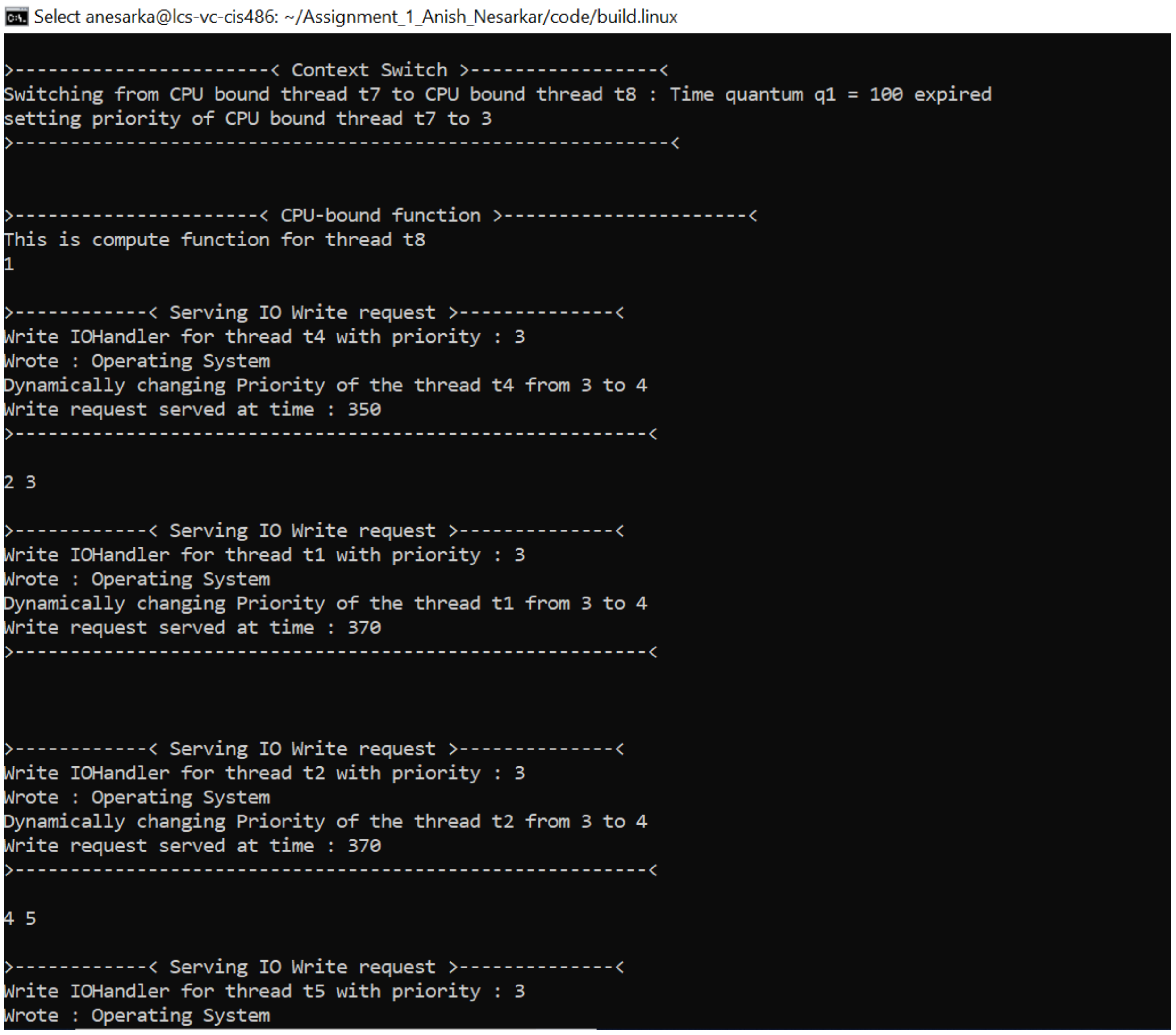
```
Select anesarka@lcs-vc-cis486: ~/Assignment_1_Anish_Nesarkar/code/build.linux
anesarka@lcs-vc-cis486:~/Assignment_1_Anish_Nesarkar/code/build.linux$ ./nachos -K

----------------------< Simulating MLFQ in Nachos >------------------------

Number of levels is MLFQ : 4
Level 1 has priority 4 with Quantum q1 = 100
Level 2 has priority 3 with Quantum q2 = 200
Level 3 has priority 2 with Quantum q3 = 300
Level 4 has priority 1 with Quantum q4 = 400

The number threads forked for the simulation : 12

Number of IO-bound threads : 5
Number of CPU-bound threads : 5
Number of Mixed threads : 2

>----------------------< IO-bound function >----------------------<
This is IO function for Thread t1


>--------------------------< Register Write >-----------------------<
Register Write with Thread ID: 1
The Write request will be served after the time : 362
The priority of the thread is initially set to : 4
>------------------------------------------------------------------<



>-----------------------< Context Switch >-----------------<
Switching from IO bound thread t1 to IO bound thread t2 : Time quantum q1 = 100 expired
setting priority of IO bound thread t1 to 3
>----------------------------------------------------------<


>----------------------< IO-bound function >----------------------<
This is IO function for Thread t2


>--------------------------< Register Write >-----------------------<
Register Write with Thread ID: 2
The Write request will be served after the time : 362
```

- The above out shows spawning of 12 threads of which 5 are CPU bound, 5 are IO bound and 2 are hybrid.
- It shows MLFQ design parameters
- It shows registering an IO bound request in the IOInterrupt and sets the timer according to the delay.
- It shows context switching from one thread to another after time quantum q1 has expired.

2. IO Read and Write Request


anesarka@lcs-vc-cis486: ~/Assignment_1_Anish_Nesarkar/code/build.linux

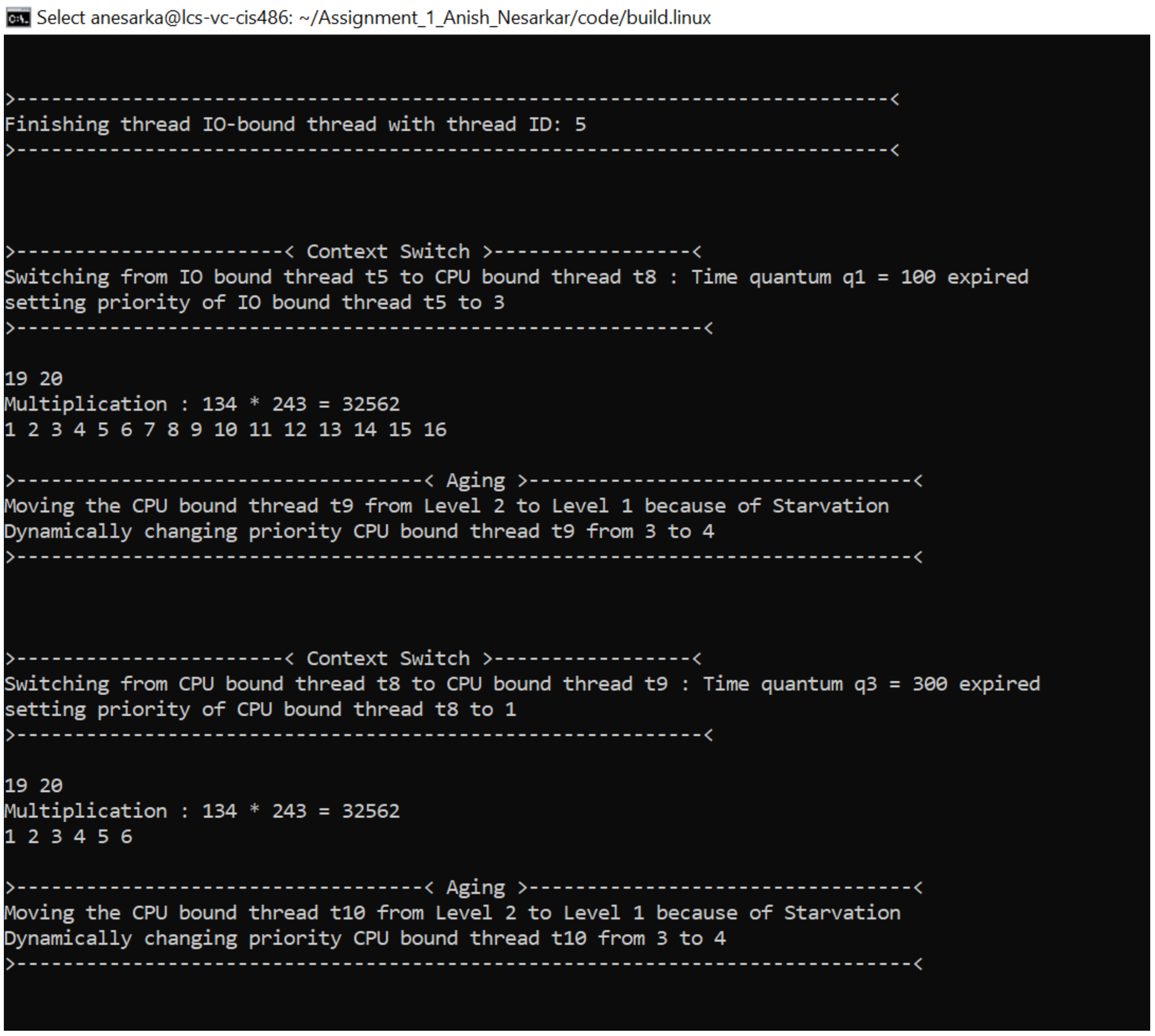
```
10 11 12 13 14 15 16 17 18

>-----------------------< Context Switch >-----------------<
Switching from CPU bound thread t10 to Hybrid thread t11 : Time quantum q2 = 200 expired
setting priority of CPU bound thread t10 to 2
>------------------------------------------------------------<

10 11 12 13 14 15 16 17 18

>-----------------------< Context Switch >-----------------<
Switching from Hybrid thread t11 to Hybrid thread t12 : Time quantum q2 = 200 expired
setting priority of Hybrid thread t11 to 2
>------------------------------------------------------------<



>------------< Serving IO Read request >--------------<
Read IOHandler for thread t1 with priority : 3
Information Read : A Read string
Dynamically changing Priority of the thread t1 from 3 to 4
Read request served at time : 1440
>----------------------------------------------------------<

10 11

>------------< Serving IO Read request >--------------<
Read IOHandler for thread t2 with priority : 3
Information Read : A Read string
Dynamically changing Priority of the thread t2 from 3 to 4
Read request served at time : 1460
>----------------------------------------------------------<

12 13
```



```
Select anesarka@lcs-vc-cis486: ~/Assignment_1_Anish_Nesarkar/code/build.linux
>-----------------------< Context Switch >-----------------<
Switching from CPU bound thread t7 to CPU bound thread t8 : Time quantum q1 = 100 expired
setting priority of CPU bound thread t7 to 3
>-----------------------------------------------------------<


>----------------------< CPU-bound function >----------------------<
This is compute function for thread t8
1

>------------< Serving IO Write request >--------------<
Write IOHandler for thread t4 with priority : 3
Wrote : Operating System
Dynamically changing Priority of the thread t4 from 3 to 4
Write request served at time : 350
>---------------------------------------------------------<

2 3

>------------< Serving IO Write request >--------------<
Write IOHandler for thread t1 with priority : 3
Wrote : Operating System
Dynamically changing Priority of the thread t1 from 3 to 4
Write request served at time : 370
>---------------------------------------------------------<



>------------< Serving IO Write request >--------------<
Write IOHandler for thread t2 with priority : 3
Wrote : Operating System
Dynamically changing Priority of the thread t2 from 3 to 4
Write request served at time : 370
>---------------------------------------------------------<

4 5

>------------< Serving IO Write request >--------------<
Write IOHandler for thread t5 with priority : 3
Wrote : Operating System
```

- It above output shows Serving an IO request when its time has arrived while other threads were running.
- If it is a Write IO operation, the string that is written is displayed and if it read Io operation, the string that is stored in buffer is displayed.
- The dynamically changing priority is displayed for the IO operation.

3. Aging

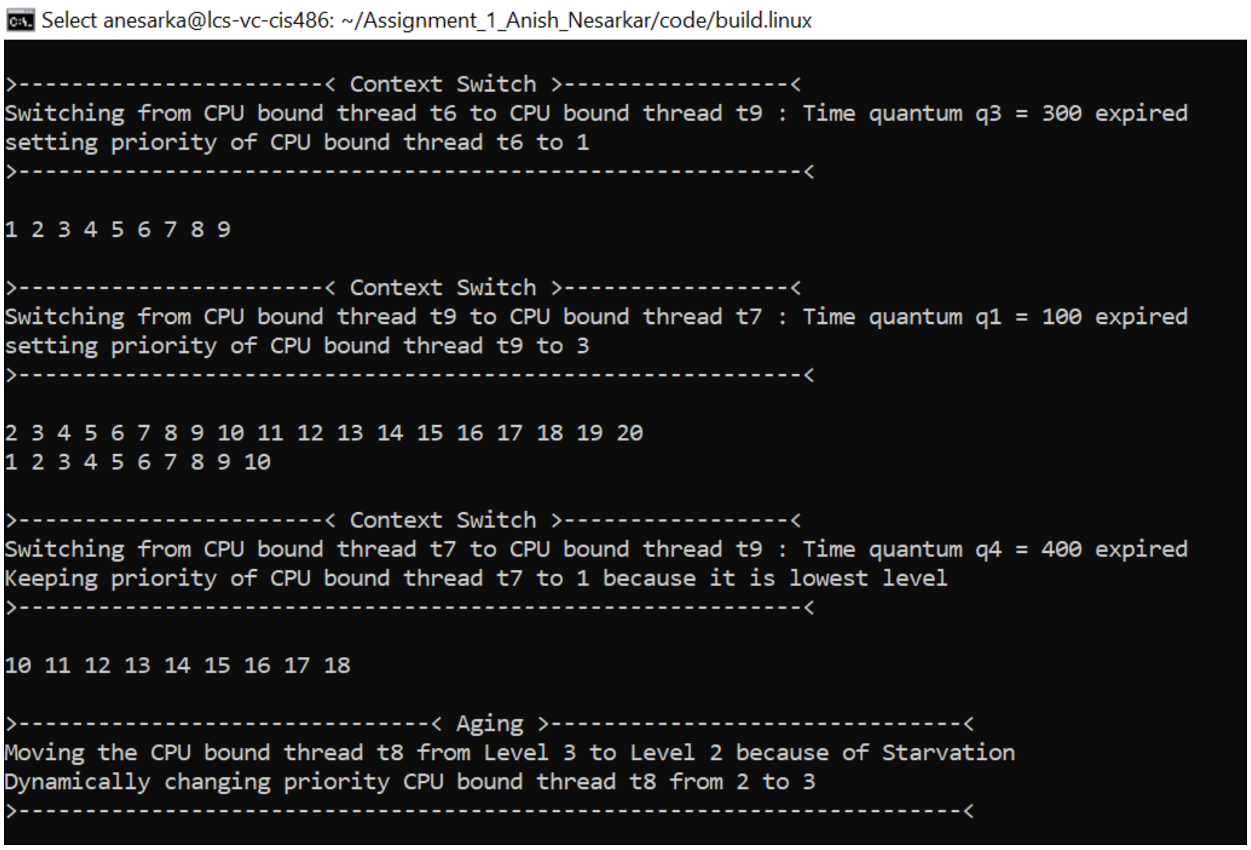


Select anesarka@lcs-vc-cis486: ~/Assignment_1_Anish_Nesarkar/code/build.linux

```
>----------------------------------------------------------------------------<
Finishing thread IO-bound thread with thread ID: 5
>----------------------------------------------------------------------------<



>-----------------------< Context Switch >-----------------<
Switching from IO bound thread t5 to CPU bound thread t8 : Time quantum q1 = 100 expired
setting priority of IO bound thread t5 to 3
>------------------------------------------------------------<

19 20
Multiplication : 134 * 243 = 32562
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

>-----------------------------------< Aging >--------------------------------<
Moving the CPU bound thread t9 from Level 2 to Level 1 because of Starvation
Dynamically changing priority CPU bound thread t9 from 3 to 4
>------------------------------------------------------------------------------<



>-----------------------< Context Switch >-----------------<
Switching from CPU bound thread t8 to CPU bound thread t9 : Time quantum q3 = 300 expired
setting priority of CPU bound thread t8 to 1
>------------------------------------------------------------<

19 20
Multiplication : 134 * 243 = 32562
1 2 3 4 5 6

>-----------------------------------< Aging >--------------------------------<
Moving the CPU bound thread t10 from Level 2 to Level 1 because of Starvation
Dynamically changing priority CPU bound thread t10 from 3 to 4
>------------------------------------------------------------------------------<
```

- The above output displays aging.
- When the waiting time of the thread in any level queue is greater than a threshold value, the thread is moved up to the above level and its priority is changed dynamically to higher number.

4. Level 4



Select anesarka@lcs-vc-cis486: ~/Assignment_1_Anish_Nesarkar/code/build.linux

```
>----------------------< Context Switch >----------------<
Switching from CPU bound thread t6 to CPU bound thread t9 : Time quantum q3 = 300 expired
setting priority of CPU bound thread t6 to 1
>-------------------------------------------------------<

1 2 3 4 5 6 7 8 9

>----------------------< Context Switch >----------------<
Switching from CPU bound thread t9 to CPU bound thread t7 : Time quantum q1 = 100 expired
setting priority of CPU bound thread t9 to 3
>-------------------------------------------------------<

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10

>----------------------< Context Switch >----------------<
Switching from CPU bound thread t7 to CPU bound thread t9 : Time quantum q4 = 400 expired
Keeping priority of CPU bound thread t7 to 1 because it is lowest level
>-------------------------------------------------------<

10 11 12 13 14 15 16 17 18

>------------------------------< Aging >------------------------------<
Moving the CPU bound thread t8 from Level 3 to Level 2 because of Starvation
Dynamically changing priority CPU bound thread t8 from 2 to 3
>-------------------------------------------------------------------<
```

- The above output shows that if a thread is in level 4 after time quantum is expired, it stays in the lower level.