OS HW04 GROUP 11
311512011 楊士賢

Part 1:Implementation Detail

(1). thread.h

add features to thread(process) for later scheduling

```
// Add priority to thread (private)
    int priority;
    double burstTime;
    double predictTime;
    double remainingTime;
    double runningTime;
    double waitingTime;
    double totalWaitingTime;
```

and functions at public to update/get the features

```
int getPriority() { return priority;}
    void setPriority(int p) { priority = p; }

    double getBurstTime(){ return burstTime; }
    void setBurstTime(double t) { burstTime = t; }

    double getPredictTime(){ return predictTime; }
    void setPredictTime(double t) {predictTime = t; }

    double getRemainingTime(){ return remainingTime; }
    void setRemainingTime(double t){ remainingTime = t; }

    double getRunningTime(){ return runningTime; }
    void setRunningTime(double t) { runningTime=t; }

    double getWaitingTime(){ return waitingTime; }
    void setWaitingTime(double t){ waitingTime = t;}

    double getTotalWaitingTime(){ return totalWaitingTime; }
    void setTotalWaitingTime(double t) { totalWaitingTime = t;}

    int getLevel() { return 3-priority / 50;}
```

(2). thread.cc

at function Yield(): when thread is in Ready to Run state, update BurstTime and RemainingTime

```
void Thread::Yield ()
{
    Thread *nextThread;
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);

    ASSERT(this == kernel->currentThread);

    DEBUG(dbgThread, "Yielding thread: " << name);

    double time = (double)kernel->stats->totalTicks - getRunningTime();
    setBurstTime(getBurstTime()+time);
    setRemainingTime(getRemainingTime()+time);


    kernel->scheduler->ReadyToRun(this);
    nextThread = kernel->scheduler->FindNextToRun();

    if (nextThread != NULL) {
        DEBUG(dbgExpr, "[E] Tick [" << kernel->stats->totalTicks
                       << "]: Thread [" << nextThread->getID() <<
                       "] is now selected for excution, thread [" <<
                       ID << "] is replaced, and it has excuted ["
                       << getBurstTime() << "] ticks");

        // if(nextThread->getPriority()>this->getPriority()){
            // kernel->scheduler->FindNextToRun();
            // kernel->scheduler->ReadyToRun(this);
            // kernel->scheduler->Run(nextThread, FALSE);
        // }
        // kernel->scheduler->ReadyToRun(this);
        kernel->scheduler->Run(nextThread, FALSE);
    }
    (void) kernel->interrupt->SetLevel(oldLevel);
}
```

- When the thread transitions to the Waiting State for completion, we adjust the BurstTime and utilize the accumulated BurstTime to estimate the duration of the next burst time (nextGuessTime). We then initialize RemainingTime to nextGuessTime. Ultimately, the thread's BurstTime is set to zero.

```
void Thread::Sleep (bool finishing)
{
    Thread *nextThread;

    ASSERT(this == kernel->currentThread);
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    DEBUG(dbgThread, "Sleeping thread: " << name);

    status = BLOCKED;
    double time = (double)kernel->stats->totalTicks - getRunningTime();
    setBurstTime(getBurstTime() - time);
    double nextPredictTime = 0.5*getBurstTime() + 0.5*getPredictTime();

    DEBUG(dbgExpr, "[D] Tick [" << kernel->stats->totalTicks <<
                   "]: Thread [" << ID << "] update approximate burst time, from: ["
                   << getPredictTime() << "], add [" << getBurstTime() << "], to ["
                   << nextPredictTime << "]");

    setPredictTime(nextPredictTime);
    setRemainingTime(nextPredictTime);
    double PrevBurstTime = getBurstTime();
    setBurstTime(0);

    // kernel->scheduler->ReadyToRun(this);
    while ((nextThread = kernel->scheduler->FindNextToRun()) == NULL) {
        kernel->interrupt->Idle();  // no one to run, wait for an interrupt
    }

    // returns when it's time for us to run
    DEBUG(dbgExpr, "[E] Tick [" << kernel->stats->totalTicks << "]: Thread ["
                        << nextThread->getID() << "] is now selected for excution, thread
["
                        << ID << "] is replaced, and it has excuted [" << PrevBurstTime <<
"] ticks");

    // returns when it's time for us to run
    kernel->scheduler->Run(nextThread, finishing);
}
```

(3). scheduler.h

```
    SortedList<Thread *> *L1queue; // premeptive SJF
    SortedList<Thread *> *L2queue; // non-preemptive priority
    List<Thread *> *L3queue; // RR
```

- New 3 queue for 3 different scheduling strategy.L1:Shortest Job First,L2: non-preemptive priority, L3: Round Robin

(4). Scheduler.cc

Implement the strategy for three queue，SJF(compare the job time)，priority(sort the priority)

```
int compare_priority(Thread *t1, Thread *t2){
    // cout<<"t1:"<<t1->getPriority()<<" || t2:"<<t2->getPriority()<<"\n";
    if(t1->getPriority() > t2->getPriority()){
        return -1;
    }else if(t1->getPriority() < t2->getPriority()){
        return 1;
    }else{
        return (t1->getID() < t2->getID())?-1:1;
    }
    return 0;
}

int compare_remainingTime(Thread *t1, Thread *t2){
    if (t1->getRemainingTime() == t2->getRemainingTime()) return 0;
    return (t1->getRemainingTime() > t2->getRemainingTime()) ? 1 : -1;
}


List<Thread *> *
Scheduler::getQueue(int level){
    if(level==1) return L1queue;
    if(level==2) return L2queue;
    return L3queue;
}
```

- At the scheduler, we initialize three queues, for the level 1 queue and level 2 queue, we set them as a sorted list, as soon as one thread is inserted into the queue, the sorting algorithm will activate automatically.

```
//----------------------------------------------------------------------
// Scheduler::Scheduler
//  Initialize the list of ready but not running threads.
//  Initially, no ready threads.
//----------------------------------------------------------------------

Scheduler::Scheduler()
{
```

```
    // readyList = new List<Thread *>;
    // priorityList = new SortedList<Thread *>(compare_priority);

    L1queue = new SortedList(compare_remainingTime);
    L2queue = new SortedList(compare_priority);
    L3queue = new List<Thread *>;
    toBeDestroyed = NULL;
}
```

- Mark down the start waiting time when the thread enters the ready queue.

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

    thread->setWaitingTime(kernel->stats->totalTicks);
    int level = thread->getLevel();
    if(level==1){
        L1queue->Insert(thread);
    }else if(level==2){
        L2queue->Insert(thread);
    }
    thread->setStatus(READY);
    DEBUG(dbgExpr, "[A] Tick ["<<kernel->stats->totalTicks<<"]: Thread ["
                    << thread->getID() << "] is inserted into queue L["<<level<<"]");

    // priorityList->Insert(thread);
    // priorityList->Apply(ThreadPrint);

    // readyList->Append(thread);
}
```

- If there is a thread waiting to run, pop out the thread from the queue and then run the thread.

```
//----------------------------------------------------------------------
// Scheduler::FindNextToRun
//  Return the next thread to be scheduled onto the CPU.
//  If there are no ready threads, return NULL.
// Side effect:
//  Thread is removed from the ready list.
//----------------------------------------------------------------------

Thread *
Scheduler::FindNextToRun ()
{
```

```
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    int level;

    Thread *nextThread = NULL;

    if(!L1queue->IsEmpty()){
        nextThread = L1queue->RemoveFront();
        level = 1;
    }else if(!L2queue->IsEmpty()){
        nextThread = L2queue->RemoveFront();
        level = 2;
    }else if(!L3queue->IsEmpty()){
        nextThread = L3queue->RemoveFront();
        level = 3;
    }else{
        return NULL;
    }
    DEBUG(dbgExpr, "[B] Tick [" << kernel->stats->totalTicks <<
                        "]: Thread [" << nextThread->getID() <<
                        "] is removed from queue L[" << level << "]");
    return nextThread;
    // if (readyList->IsEmpty()) {
    //  return NULL;
    // } else {
    //  return readyList->RemoveFront();
    // }
}
```

- Update next thread's startTunning time before switching and set wait time=0.

```
void Scheduler::Run (Thread *nextThread, bool finishing)
{
    Thread *oldThread = kernel->currentThread;
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    if (finishing) {    // mark that we need to delete current thread
        ASSERT(toBeDestroyed == NULL);
        toBeDestroyed = oldThread;
    }
    if (oldThread->space != NULL) { // if this thread is a user program,
        oldThread->SaveUserState();     // save the user's CPU registers
        oldThread->space->SaveState();
    }
    oldThread->CheckOverflow();         // check if the old thread
```

```
                                    // had an undetected stack overflow
    kernel->currentThread = nextThread;  // switch to the next thread
    nextThread->setStatus(RUNNING);      // nextThread is now running

    DEBUG(dbgThread, "Switching from: " << oldThread->getName() << " to: " <<
nextThread->getName());

    nextThread->setRunningTime(kernel->stats->totalTicks); //update running timer
    nextThread->setTotalWaitingTime(0);
    // This is a machine-dependent assembly language routine defined
    // in switch.s.  You may have to think
    // a bit to figure out what happens after this, both from the point
    // of view of the thread and from the perspective of the "outside world".

    SWITCH(oldThread, nextThread);
    // we're back, running oldThread
    // interrupts are off when we return from switch!
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    DEBUG(dbgThread, "Now in thread: " << oldThread->getName());

    CheckToBeDestroyed();       // check if thread we were running
                    // before this one has finished
                    // and needs to be cleaned up
    if (oldThread->space != NULL) {     // if there is an address space
        oldThread->RestoreUserState();     // to restore, do it.
        oldThread->space->RestoreState();
    }
}
```

(5). Alarm.h

Add aging algorithm in case of starvation.

```
class Alarm : public CallBackObj {
  public:
    Alarm(bool doRandomYield);  // Initialize the timer, and callback
        // to "toCall" every time slice.
    ~Alarm() { delete timer; }
    void WaitUntil(int x);  // suspend execution until time > now + x
                            // this method is not yet implemented
    void Aging();
  private:
    Timer *timer;   // the hardware timer device
    void CallBack();    // called when the hardware
```

```
        // timer generates an interrupt
};
```

(6). Alarm.cc

Add an if else statement to judge whether it is preemptive or non-preemptive using the level of thread， only level 2(priority scheduling) is non-preemptive

```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();

    int level = kernel->currentThread->getLevel();

    if(status != IdleMode){
        Aging();
        if(level == 2){
            if(!(kernel->scheduler->getQueue(1)->IsEmpty())){
                interrupt->YieldOnReturn();
            }
        }else{
            interrupt->YieldOnReturn();
        }
    }
}
```

1. The Aging function iterates over three priority levels (queues) labeled 1 to 3.
2. For each priority level, it retrieves the list of threads in that priority queue.
3. It then iterates through the threads in the queue using a ListIterator.
4. For each thread, it updates the total waiting time, calculates the time the thread has been waiting, and checks if the waiting time exceeds a threshold (1500 ticks).
5. If the waiting time is greater than or equal to 1500 ticks, it performs the aging process.
   - It increases the thread's priority by a fixed amount (up to a maximum of 149).
   - It subtracts 1500 from the total waiting time to prevent continuous aging.
   - It checks if the thread's new priority level is lower than the current priority queue (i.e., nextLevel < i).
6. If the thread's new priority level is lower than the current priority queue, it removes the thread from the current queue and inserts it into the lower-priority queue.

```
void
Alarm::Aging(){
    for(int i=1;i<=3;i++){
        List<Thread *>*queue = kernel->scheduler->getQueue(i);
        ListIterator<Thread *> *it = new ListIterator<Thread *>(queue);
```

```
        double time;
        int prevPriority;
        int nextLevel;
        for(;!it->IsDone();it->Item()){
            Thread *thread = it->Item();
            thread->setTotalWaitingTime(thread->getTotalWaitingTime() +
                                kernel->stats->totalTicks - thread->getWaitingTime());

            thread->setWaitingTime(kernel->stats->totalTicks);
            time = thread->getTotalWaitingTime();

            //aging
            if(time>=1500){
                prevPriority = thread->getPriority();
                thread->setPriority(min(prevPriority + 10, 149));
                thread->setTotalWaitingTime(thread->getTotalWaitingTime() - 1500);
                nextLevel = thread->getLevel();
                DEBUG(dbgExpr, "[C] Tick [" << kernel->stats->totalTicks <<
                                "]: Thread [" << thread->getID() <<
                                "] change its priority from [" <<
                                prevPriority << "] to [" <<
                                thread->getPriority() << "]");
                if(nextLevel<i){
                    DEBUG(dbgExpr, "[B] Tick [" << kernel->stats->totalTicks <<
                                "]: Thread [" << thread->getID() <<
                                "] is removed from queue L[" << i << "]");
                    queue->Remove(thread);
                    DEBUG(dbgExpr, "[A] Tick [" << kernel->stats->totalTicks <<
                                "]: Thread [" << thread->getID() <<
                                "] is inserted into queue L[" << nextLevel << "]");
                    ((SortedList<Thread *> *)kernel->scheduler->getQueue(nextLevel))-
>Insert(thread);
                }
            }
        }
    }
}
```

Result:

Running L3 test_1

timeout 1 ../build.linux/nachos -ep hw4_normal_test1 0 -ep hw4_normal_test2 0

```
Running L3 test_1
1
1
2
2
2
2
2
1
1
1
2
2
2
2
2
1hw4_normal_test1 || Priority = 0
hw4_normal_test2 || Priority = 0
return value:0

1
1
1
1
return value:0
L3 test_1 done
```

- 這邊系統的 Round Robing 預設的 time slice 表現比較奇怪，目前還沒檢查出來到底是什麼情況，理論上應該是每個 thread 平均分配 CPU 的 time slice 才對。

Running L2 test_1

timeout 1 ../build.linux/nachos -ep hw4_normal_test1 50 -ep hw4_normal_test2 50

```
Running L2 test_1
1
1
1
1
1
1
1
1
1
1
hw4_normal_test1 || Priority = 50
hw4_normal_test2 || Priority = 50
return value:0
2
2
2
2
2
2
2
2
2
2
return value:0
Running L2 test_1 done
```

Running L2 test_2

timeout 1 ../build.linux/nachos -ep hw4_normal_test1 50 -ep hw4_normal_test2 90

```
Running L2 test_2
2
2
2
2
2
2
2
2
2
2
hw4_normal_test1 || Priority = 50
hw4_normal_test2 || Priority = 90
return value:0
1
1
1
1
1
1
1
1
1
1
return value:0
Running L2 test_2 done
```

Running L2 test_3

timeout 1 ../build.linux/nachos -ep hw4_normal_test1 70 -ep hw4_normal_test2 80 -ep hw4_normal_test3 50

```
Running L2 test_3
2
2
2
2
2
2
2
2
2
2
hw4_normal_test1 || Priority = 70
hw4_normal_test2 || Priority = 80
hw4_normal_test3 || Priority = 50
return value:0
1
1
1
1
1
1
1
1
1
1
return value:0
3
3
3
3
3
3
3
3
3
3
return value:0
Running L2 test_3 done
```

Running L1 test_1

timeout 1 ../build.linux/nachos -ep hw4_normal_test1 100 -ep hw4_normal_test2 100

```
Running L1 test_1
1
1
1
1
1
1
1
1
1
1
hw4_normal_test1 || Priority = 100
hw4_normal_test2 || Priority = 100
return value:0
2
2
2
2
2
2
2
2
2
2
return value:0
Running L1 test_1 done
```

Running L1 test_2

timeout 3 ../build.linux/nachos -ep hw4_delay_test1 100 -ep hw4_normal_test2 100

- 這邊因為 thread 1 的 remaining time 比較長，因此中途被 thread 2 搶占

```
Running L1 test_2
1
1
2
2
2
2
2
2
2
2
2
2
hw4_delay_test1 || Priority = 100
hw4_normal_test2 || Priority = 100
return value:0
1
1
1
1
1
1
1
1
return value:0
Running L1 test_2 done
```

Running Aging L3 -> L2 test_1

timeout 3 ../build.linux/nachos -ep hw4_delay_test1 50 -ep hw4_normal_test2 45

```
Running Aging L3 -> L2 test_1
1
1
1
1
1
1
1
1
1
1
hw4_delay_test1 || Priority = 50
hw4_normal_test2 || Priority = 45
return value:0
2
2
2
2
2
2
2
2
2
2
return value:0
Running Aging L3 -> L2 test_1 done
================================================
```

Part 2:Contribution

1. **Describe details and percentage of each member’s contribution.**