110705017 何翊華 110705063 廖偉辰

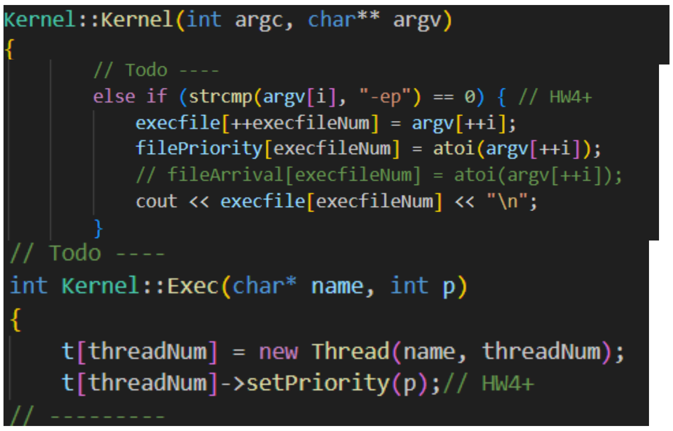
Part 1:Implementation Detail

Priority Setting:

- Add new command line identifier **-ep**, and add list **filePriority[10]** to store the priority value
- Add the variable **int priority** to thread to store the value
- Add functions **setPriority, getPriority** to set priority when executing in kernel
- Add new parameter **int p** of Exec() in kernel to pass the priority value

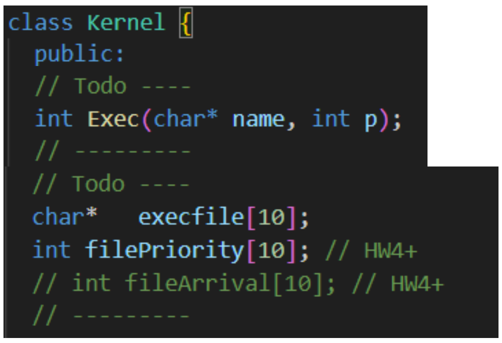
The following are modifications:

threads/kernel.cc



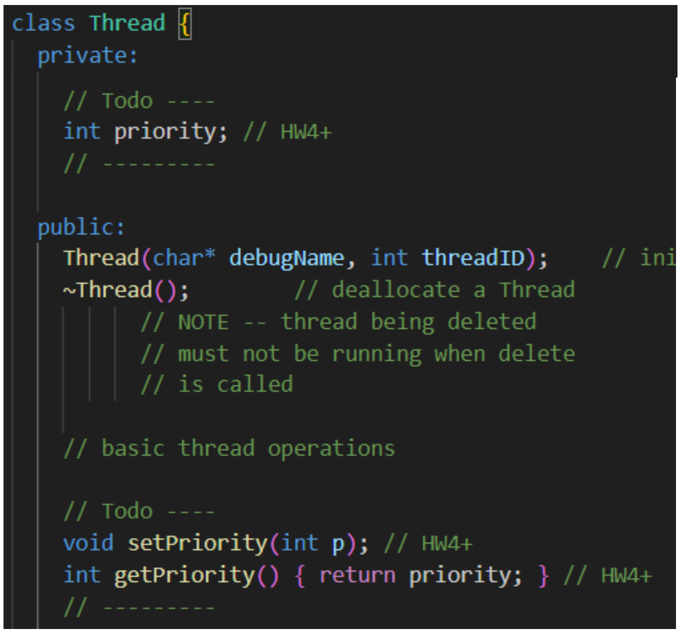
```
Kernel::Kernel(int argc, char** argv)
{
        // Todo ----
        else if (strcmp(argv[i], "-ep") == 0) { // HW4+
            execfile[++execfileNum] = argv[++i];
            filePriority[execfileNum] = atoi(argv[++i]);
            // fileArrival[execfileNum] = atoi(argv[++i]);
            cout << execfile[execfileNum] << "\n";
        }
// Todo ----
int Kernel::Exec(char* name, int p)
{
    t[threadNum] = new Thread(name, threadNum);
    t[threadNum]->setPriority(p);// HW4+
// ---------
```

threads/kernel.h



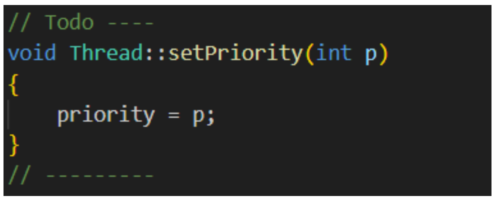
```
class Kernel {
  public:
  // Todo ----
  int Exec(char* name, int p);
  // ---------
  // Todo ----
  char*   execfile[10];
  int filePriority[10]; // HW4+
  // int fileArrival[10]; // HW4+
  // ---------
```

threads/thread.h



```
class Thread {
  private:

    // Todo ----
    int priority; // HW4+
    // ---------

  public:
    Thread(char* debugName, int threadID);     // ini
    ~Thread();          // deallocate a Thread
          // NOTE -- thread being deleted
          // must not be running when delete
          // is called

    // basic thread operations

    // Todo ----
    void setPriority(int p); // HW4+
    int getPriority() { return priority; } // HW4+
    // ---------
```

threads/thread.cc

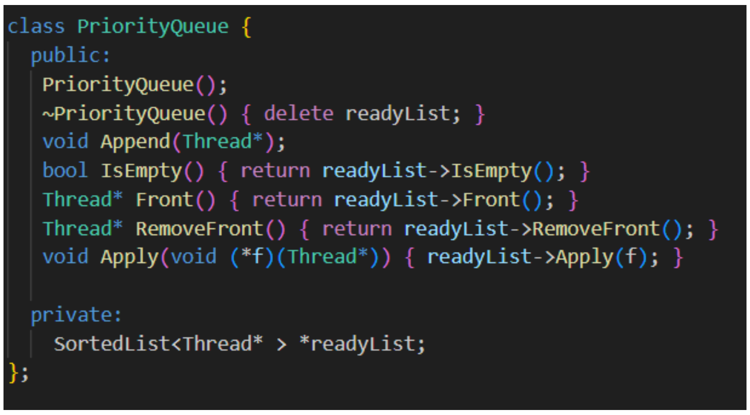


```
// Todo ----
void Thread::setPriority(int p)
{
    priority = p;
}
// ---------
```

Priority Queue Scheduler:

We add **Class PriorityQueue** as our scheduler, which is similar to the original ready queue method, however ***readyList** is replaced by **SortedList<Thread* > *readyList** in our implementation for the sake of sorting threads by their priority value.

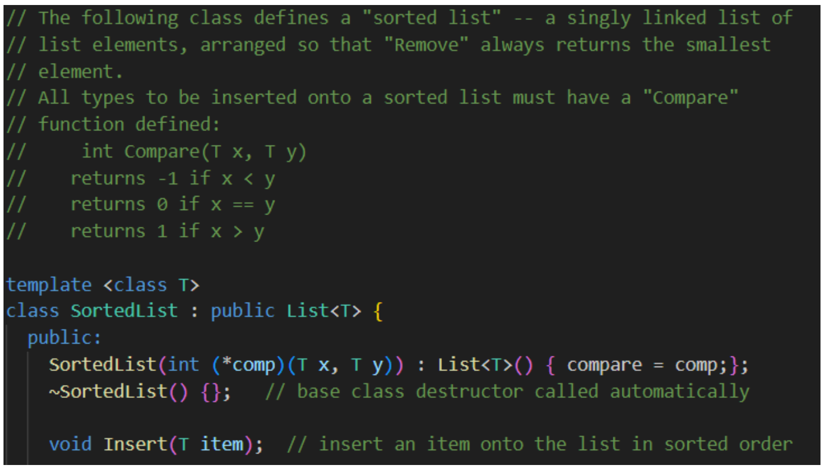
threads/scheduler.h



```
class PriorityQueue {
  public:
   PriorityQueue();
   ~PriorityQueue() { delete readyList; }
   void Append(Thread*);
   bool IsEmpty() { return readyList->IsEmpty(); }
   Thread* Front() { return readyList->Front(); }
   Thread* RemoveFront() { return readyList->RemoveFront(); }
   void Apply(void (*f)(Thread*)) { readyList->Apply(f); }

  private:
    SortedList<Thread* > *readyList;
};
```

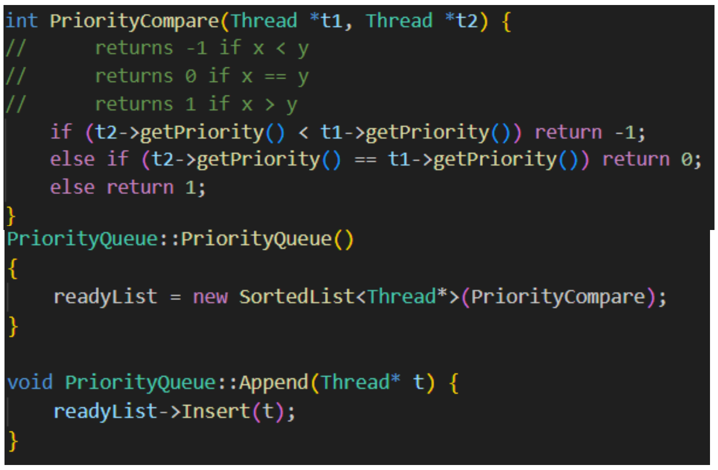
SortedList<Thread* > *readyList uses predefined data structure in lib/list, which requires the compare function as input parameter to sort the threads while the thread is inserted into the ready queue, therefore we defined our compare function in scheduler.cc directly.
Notice that we replace append method by insert method here.

lib/list.h



```
// The following class defines a "sorted list" -- a singly linked list of
// list elements, arranged so that "Remove" always returns the smallest
// element.
// All types to be inserted onto a sorted list must have a "Compare"
// function defined:
//     int Compare(T x, T y)
//    returns -1 if x < y
//    returns 0 if x == y
//    returns 1 if x > y

template <class T>
class SortedList : public List<T> {
  public:
    SortedList(int (*comp)(T x, T y)) : List<T>() { compare = comp;};
    ~SortedList() {};   // base class destructor called automatically

    void Insert(T item);  // insert an item onto the list in sorted order
```

threads/scheduler.cc

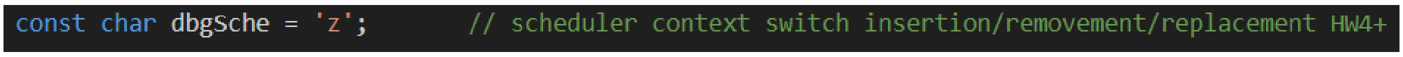


```
int PriorityCompare(Thread *t1, Thread *t2) {
//      returns -1 if x < y
//      returns 0 if x == y
//      returns 1 if x > y
    if (t2->getPriority() < t1->getPriority()) return -1;
    else if (t2->getPriority() == t1->getPriority()) return 0;
    else return 1;
}
PriorityQueue::PriorityQueue()
{
    readyList = new SortedList<Thread*>(PriorityCompare);
}

void PriorityQueue::Append(Thread* t) {
    readyList->Insert(t);
}
```

Debug Message:

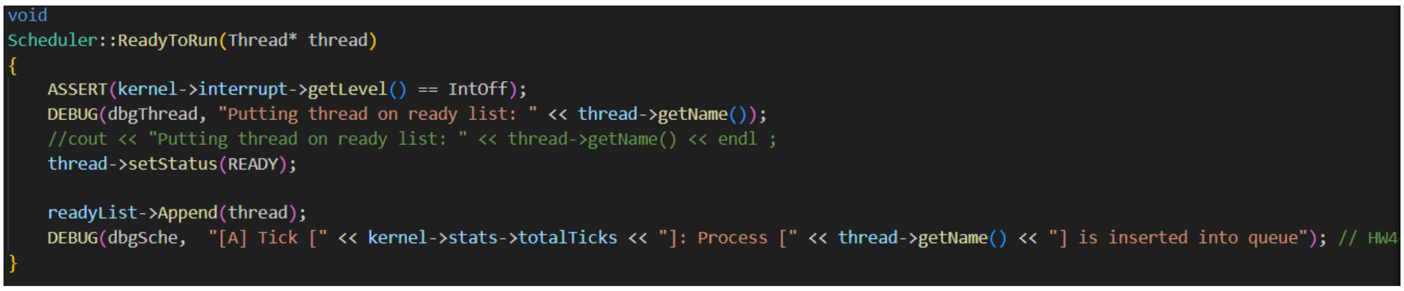
We add new debug flag **dbgSche** in debug.h and call debug when threads acting
[A] insert into queue / [B] remove from queue / [C] replace.

lib/debug.h

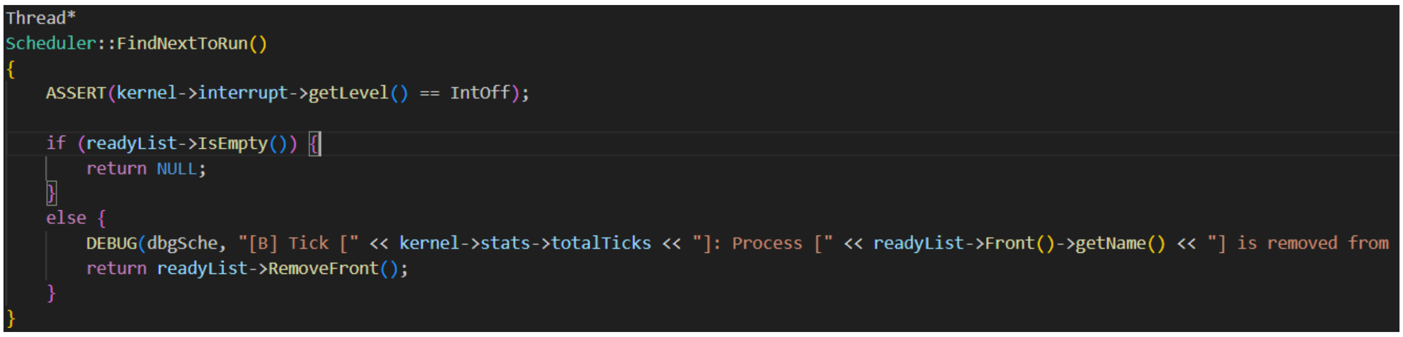


```
const char dbgSche = 'z';        // scheduler context switch insertion/removement/replacement HW4+
```

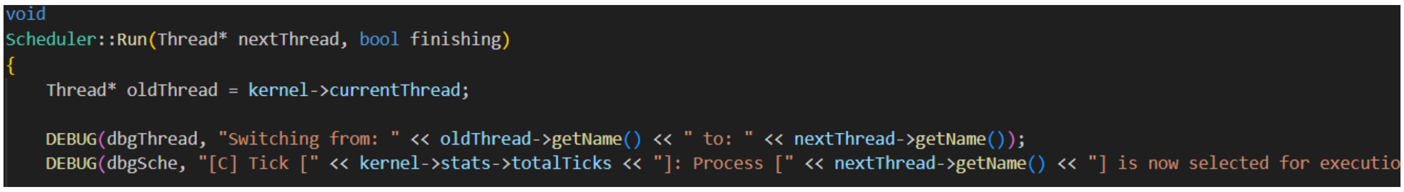
threads/scheduler.cc



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

    readyList->Append(thread);
    DEBUG(dbgSche,  "[A] Tick [" << kernel->stats->totalTicks << "]: Process [" << thread->getName() << "] is inserted into queue"); // HW4
}
```



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

    if (readyList->IsEmpty()) {
        return NULL;
    }
    else {
        DEBUG(dbgSche, "[B] Tick [" << kernel->stats->totalTicks << "]: Process [" << readyList->Front()->getName() << "] is removed from
        return readyList->RemoveFront();
    }
}
```



```
void
Scheduler::Run(Thread* nextThread, bool finishing)
{
    Thread* oldThread = kernel->currentThread;

    DEBUG(dbgThread, "Switching from: " << oldThread->getName() << " to: " << nextThread->getName());
    DEBUG(dbgSche, "[C] Tick [" << kernel->stats->totalTicks << "]: Process [" << nextThread->getName() << "] is now selected for executio
```

Part 2:Contribution

1. Describe details and percentage of each member's contribution.

姓名	負責項目	貢獻度
何翊華	Code Implement (Debug + Priority Setting) + Assist code Implement (Priority Queue Scheduler) + PPT	50%
廖偉辰	Code Implement (Priority Queue Scheduler) + Assist code Implement (Debug + Priority Setting) + PPT	50%