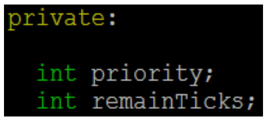
Operating System

Report for Hw2

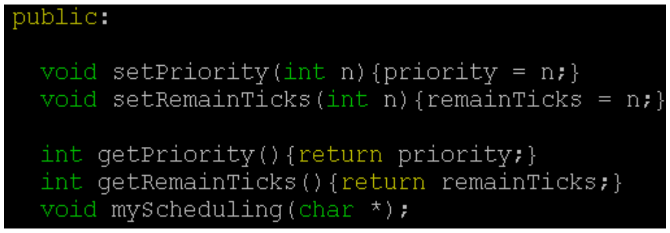
資工二 B00902008 陳明汎、B00902064 宋昊恩

1. Implementation:

First of all, we add new private member variables and public member functions in `thread.h`. Pictures captured as following:

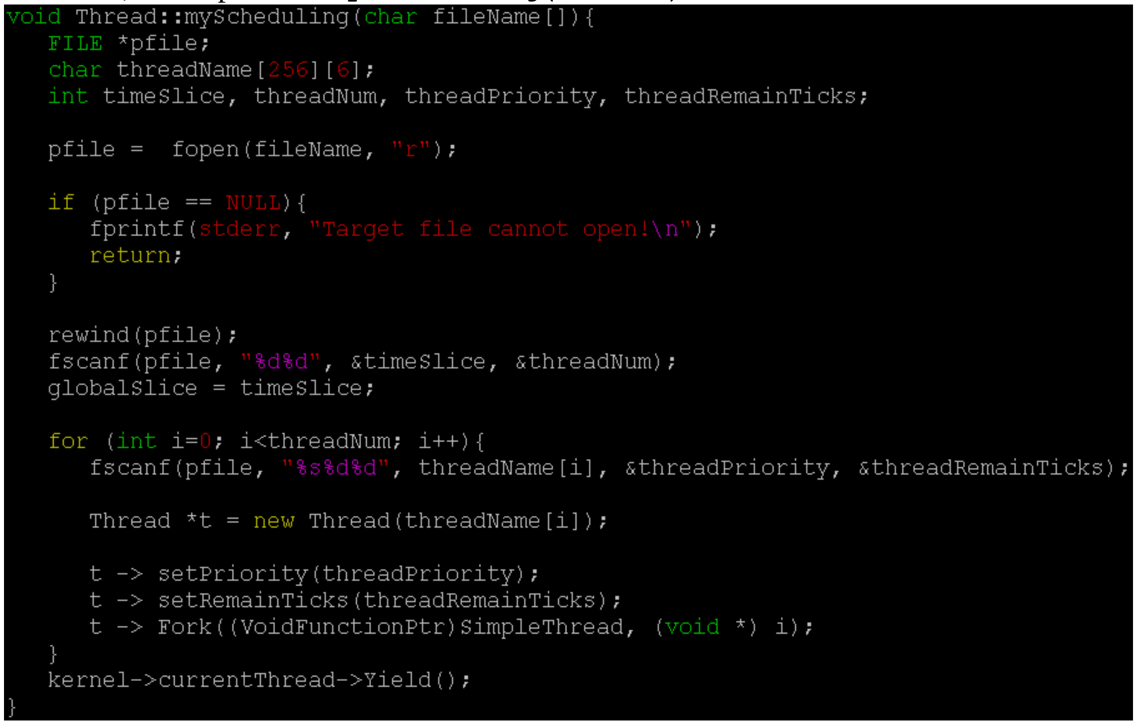


```
private:

  int priority;
  int remainTicks;
```



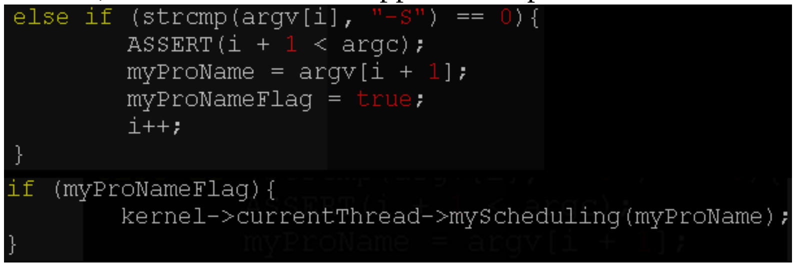
```
public:

  void setPriority(int n){priority = n;}
  void setRemainTicks(int n){remainTicks = n;}

  int getPriority(){return priority;}
  int getRemainTicks(){return remainTicks;}
  void myScheduling(char *);
```

And then, we implement `myScheduling(char *)` in `thread.cc`.



```
void Thread::myScheduling(char fileName[]){
   FILE *pfile;
   char threadName[256][6];
   int timeSlice, threadNum, threadPriority, threadRemainTicks;

   pfile =  fopen(fileName, "r");

   if (pfile == NULL){
      fprintf(stderr, "Target file cannot open!\n");
      return;
   }

   rewind(pfile);
   fscanf(pfile, "%d%d", &timeSlice, &threadNum);
   globalSlice = timeSlice;

   for (int i=0; i<threadNum; i++){
      fscanf(pfile, "%s%d%d", threadName[i], &threadPriority, &threadRemainTicks);

      Thread *t = new Thread(threadName[i]);

      t -> setPriority(threadPriority);
      t -> setRemainTicks(threadRemainTicks);
      t -> Fork((VoidFunctionPtr)SimpleThread, (void *) i);
   }
   kernel->currentThread->Yield();
}
```

Second, we made `nachos` support "`-S`" parameter in `main.cc`.

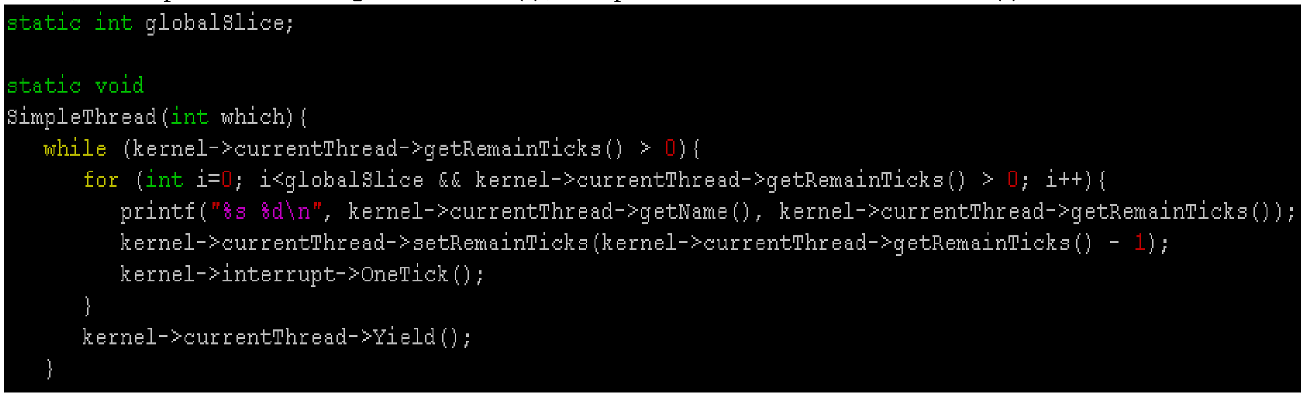


```
 else if (strcmp(argv[i], "-S") == 0){
         ASSERT(i + 1 < argc);
         myProName = argv[i + 1];
         myProNameFlag = true;
         i++;
 }
if (myProNameFlag){
        kernel->currentThread->myScheduling(myProName);
}
```

Some explanation for key changing:

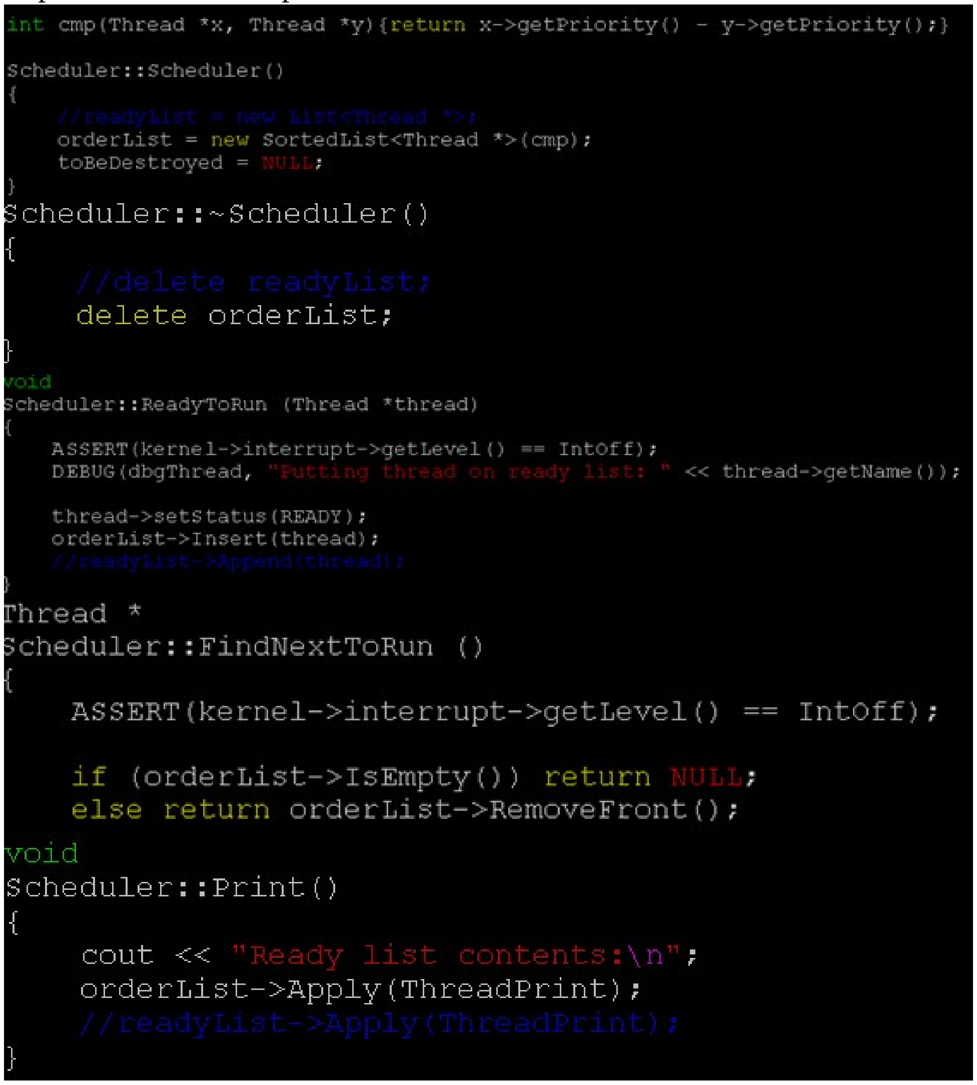
```
char *myProName = null;          // to store the file name
bool myProNameFlag = false;      // to execute myScheduling
```

Third, we implemented `SimpleThread()`, and parameter of `Thread::Fork()` in `thread.cc`.



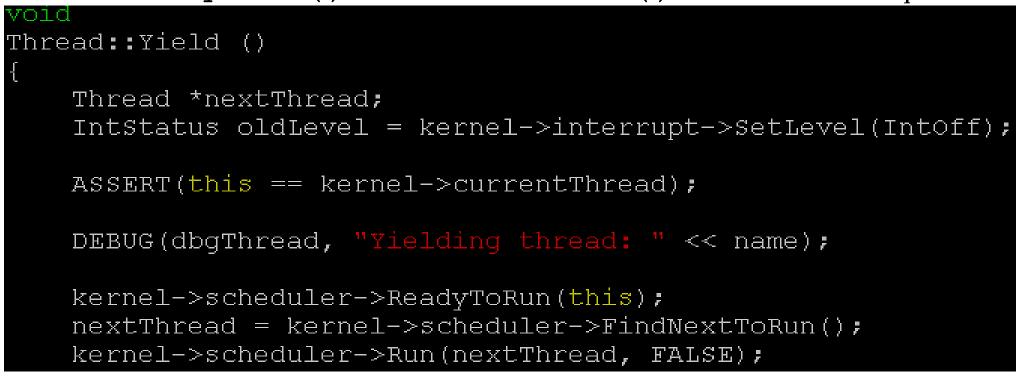
```
static int globalSlice;

static void
SimpleThread(int which){
   while (kernel->currentThread->getRemainTicks() > 0){
      for (int i=0; i<globalSlice && kernel->currentThread->getRemainTicks() > 0; i++){
         printf("%s %d\n", kernel->currentThread->getName(), kernel->currentThread->getRemainTicks());
         kernel->currentThread->setRemainTicks(kernel->currentThread->getRemainTicks() - 1);
         kernel->interrupt->OneTick();
      }
      kernel->currentThread->Yield();
   }
```

Four, we changed the data structure used for doing `Round-Robin`. Instead of using `readyList` as default, we used `SortedList` to achieve the priority requirement. We also implemented the compare function for `SortedList` in `scheduler.cc`.



```
int cmp(Thread *x, Thread *y){return x->getPriority() - y->getPriority();}

Scheduler::Scheduler()
{
    //readyList = new List<Thread *>;
    orderList = new SortedList<Thread *>(cmp);
    toBeDestroyed = NULL;
}
Scheduler::~Scheduler()
{
    //delete readyList;
    delete orderList;
}
void
Scheduler::ReadyToRun (Thread *thread)
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    DEBUG(dbgThread, "Putting thread on ready list: " << thread->getName());

    thread->setStatus(READY);
    orderList->Insert(thread);
    //readyList->Append(thread);
}
Thread *
Scheduler::FindNextToRun ()
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    if (orderList->IsEmpty()) return NULL;
    else return orderList->RemoveFront();
void
Scheduler::Print()
{
    cout << "Ready list contents:\n";
    orderList->Apply(ThreadPrint);
    //readyList->Apply(ThreadPrint);
}
```

The last modification is `Thread::Yield()` in `thread.cc`. We reordered the execution of functions `ReadyToRun()` and `FindNextToRun()` to fulfilled the requirement.



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

    ASSERT(this == kernel->currentThread);

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

    kernel->scheduler->ReadyToRun(this);
    nextThread = kernel->scheduler->FindNextToRun();
    kernel->scheduler->Run(nextThread, FALSE);
```

2. Some of our thought:

B00902008:

There are two things confused me for a long time during implementing.

For one thing, I had no idea about how `orderList` implement. I had a problem about putting elements to the right place when there were some processes with same priority. After trial and error, I got some important information.

One is that when adding an element, it will compare it with all elements in the list from the beginning of the list; another one is that when two processes with the same priority compares, it will swap with each other. Therefore, I only had to change code in the compare function.

On the other hand, there were two unknown threads existing in the list! After tracing code, I found that one is main thread and the other is for port. The solution is to initialize their priority to be smaller than any thread's priority I concerned.

After solving the problems above, I understood more about how thread scheduler do scheduling and the importance about the scheduling in the kernel.

B00902064:

This work is very interesting, and we encountered many challenges from it. After we spent about twenty hours in total, we finished our first-time code. Though everything looked perfect, it could not work at all. We then spent about four more hours trying to find out the bug, however, we failed in vain. Being depressed, we recovered to the earliest version and restarted all the jobs. Four hours later, we got our second-version code. And it did work properly!

Some points that we want to share is that actually, we did not spent time in reading the instruction in homework spec, initially. Instead, we began to trace code from instructions “`-K`”. We then knew the mechanism for `ReadyToRun()` and `FindNextToRun()`. However, we could not understand the real meaning for `oneTick()` since its implementation will be linked to `switch.S`, which we could not recognized.

That may be the main reason why, we finally found out that we didn't follow the instructions stated in homework spec, however, we still perfectly fulfilled the requirement of spec and produced the correct answer.

3. Reference:

We took little time to search for further information in this work. All the work is done by our own selves.