1. **Implement KThread.join().**

Add joinQueue to Kthread:

**private** ThreadQueue joinQueue = ThreadedKernel.*scheduler*.newThreadQueue(**true**);

Modify join(), add 6 lines:

**public** **void** join() {

Lib.*debug*(***dbgThread***, "Joining to thread: " + toString());

//thread cannot join itself

Lib.*assertTrue*(**this** != *currentThread*);

//disable interrupt

1. **boolean** intStatus = Machine.*interrupt*().disable();

//do not need to wait if thread is already finished

2. **if**(status != ***statusFinished***) {

3. joinQueue.waitForAccess(*currentThread*);

4. KThread.*sleep*();

5. }

6. Machine.*interrupt*().restore(intStatus);

}

Modify finish(),add 7 lines:

**public** **static** **void** finish() {

Lib.*debug*(***dbgThread***, "Finishing thread: " + *currentThread*.toString());

Machine.*interrupt*().disable();

Machine.*autoGrader*().finishingCurrentThread();

Lib.*assertTrue*(*toBeDestroyed* == **null**);

*toBeDestroyed* = *currentThread*;

*currentThread*.status = ***statusFinished***;

1. KThread joinThread = *currentThread*.joinQueue.nextThread();

2. **while**(joinThread != **null**)

3. {

4. joinThread.ready();

5. joinThread = *currentThread*.joinQueue.nextThread();

6. }

7. *currentThread*.joinQueue = **null**;

*sleep*();

}

1. **Implement condition variables Condition2.**

Add waitQueue to Condition2:

**private** ThreadQueue waitQueue = ThreadedKernel.*scheduler*.newThreadQueue(**true**);

Modify Sleep(), add 4 lines:

**public** **void** sleep() {

//lock is held by current thread

Lib.*assertTrue*(conditionLock.isHeldByCurrentThread());

1. **boolean** intStatus = Machine.*interrupt*().disable();

//realse lock held by current thread

conditionLock.release();

//put current thread into wait queue and go to sleep

2. waitQueue.waitForAccess(KThread.*currentThread*());

3. KThread.*sleep*();

//get lock again if waitQueue is empty

conditionLock.acquire();

4. Machine.*interrupt*().restore(intStatus);

}

Modify wake(), add 4 lines:

**public** **void** wake() {

Lib.*assertTrue*(conditionLock.isHeldByCurrentThread());

1. **boolean** intStatus = Machine.*interrupt*().disable();

2. KThread waitThread = waitQueue.nextThread();

3. **if**(waitThread!=**null**)

4. waitThread.ready();

5. Machine.*interrupt*().restore(intStatus);

}

Modify wakeAll(), add 8 lines:

**public** **void** wakeAll() {

Lib.*assertTrue*(conditionLock.isHeldByCurrentThread());

1. **boolean** intStatus = Machine.*interrupt*().disable();

2. KThread waitThread = waitQueue.nextThread();

3. **while**(waitThread!=**null**)

4. {

5. waitThread.ready();

6. waitThread = waitQueue.nextThread();

7. }

8. Machine.*interrupt*().restore(intStatus);

}

1. **Complete Alarm Class.**

Add classthreadWakeTime**:**

**public** **class** threadWakeTime {

**private** **long** wakeTime;

**private** KThread thread;

/\*\*\*\*\*\*\*\*

\* create a threadWakeTime which inludes the wake time for one thread

\* **@param** time

\* **@param** newThread

\*/

threadWakeTime(**long** time, KThread newThread) {

wakeTime = time;

thread = newThread;

}

**public** **long** getWakeTime() {

**return** wakeTime;

}

**public** KThread getThread() {

**return** thread;

}

}

Add waitlist to Alarm:

**private** LinkedList<threadWakeTime> waitList = **new** LinkedList<threadWakeTime>();

Modify waitUntil();

**public** **void** waitUntil(**long** x) {

**boolean** intStatus = Machine.*interrupt*().disable();

**long** wakeTime = Machine.*timer*().getTime() + x;

threadWakeTime thisThread

= **new** threadWakeTime(wakeTime,KThread.*currentThread*());

//insert this thread to wait list in ascending order

**int** size = waitList.size();

**int** i=0;

**for**(i=0;i<size;i++)

{

**if**((wakeTime < waitList.get(i).getWakeTime()))

**break**;

}

**if**(i>=size)

waitList.add(thisThread);

**else**

waitList.add(i,thisThread);

KThread.*sleep*();

Machine.*interrupt*().restore(intStatus);

}

Modify timerInterrupt():

**public** **void** timerInterrupt() {

**boolean** intStatus = Machine.*interrupt*().disable();

**if**(waitList != **null**)

{

**long** currentTime = Machine.*timer*().getTime();

**int** size = waitList.size();

**int** i = 0;

//wake the threads with wake time smaller than or equal to current time

**for**(i=0; i<size; i++)

{

**if**(waitList.get(i).getWakeTime() <= currentTime)

{

KThread thread = waitList.get(i).getThread();

thread.ready();

}

**else**

**break**;

}

//remove the wakeup threads from linkedlist

**for**(**int** j=i-1;j>=0;j--)

waitList.remove(j);

//Relinquish the CPU

KThread.*currentThread*().*yield*();

}

Machine.*interrupt*().restore(intStatus);

}

1. **Implement Communicator Class.**

Add size to Condition2 to get the length of waitQueue

Add variables to Communicator:

**private** Lock lock;

**private** **static** LinkedList<Integer> *wordsQueue*

= **new** LinkedList<Integer>();//Store words

Condition2 speaker;

Condition2 listener;

Modify speak():

**public** **void** speak(**int** word) {

**boolean** intStatus = Machine.*interrupt*().disable();

lock.acquire();

*wordsQueue*.add(word);

**if**(listener.size()==0){

speaker.sleep();

}

**else**{

listener.wake();

speaker.size();

}

lock.release();

Machine.*interrupt*().restore(intStatus);

}

Modify listen();

**public** **int** listen() {

**boolean** intStatus = Machine.*interrupt*().disable();

lock.acquire();

**if**(speaker.size()==0){

listener.sleep();

}

**else** {

speaker.wake();

listener.sleep();

}

lock.release();

Machine.*interrupt*().restore(intStatus);

**return** *wordsQueue*.poll();

}