Project 2 Tutorial

Threads

All Nachos threads are instances of KThread

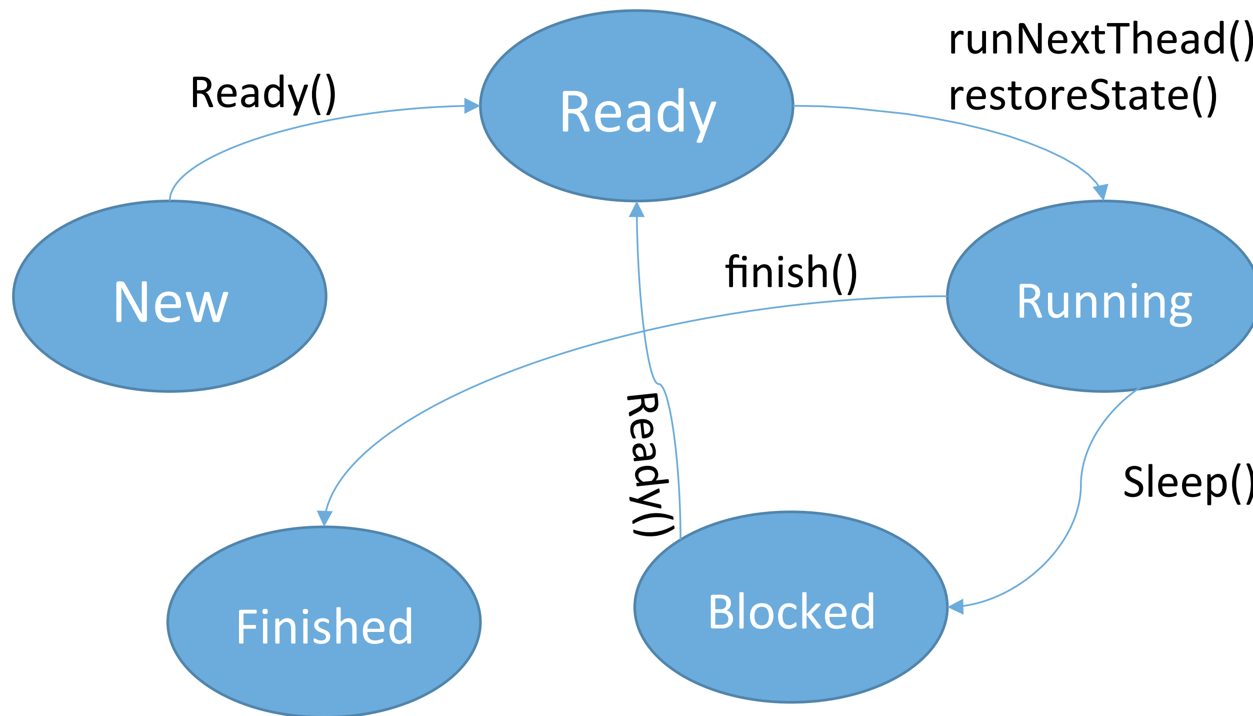
Every KThread has a status member:

- statusNew
- statusReady
- statusRunning
- statusBlocked
- statusFinished

Nachos implements threading using a java thread for each TCB object.

- Threads are synchronized. (one running at a time)

Thread Execution Cycle



Thread Execution Cycle

New: Newly created thread, yet to be forked.

Ready: Waiting for access to CPU

- Thread.ready()

Running: Thread currently using CPU

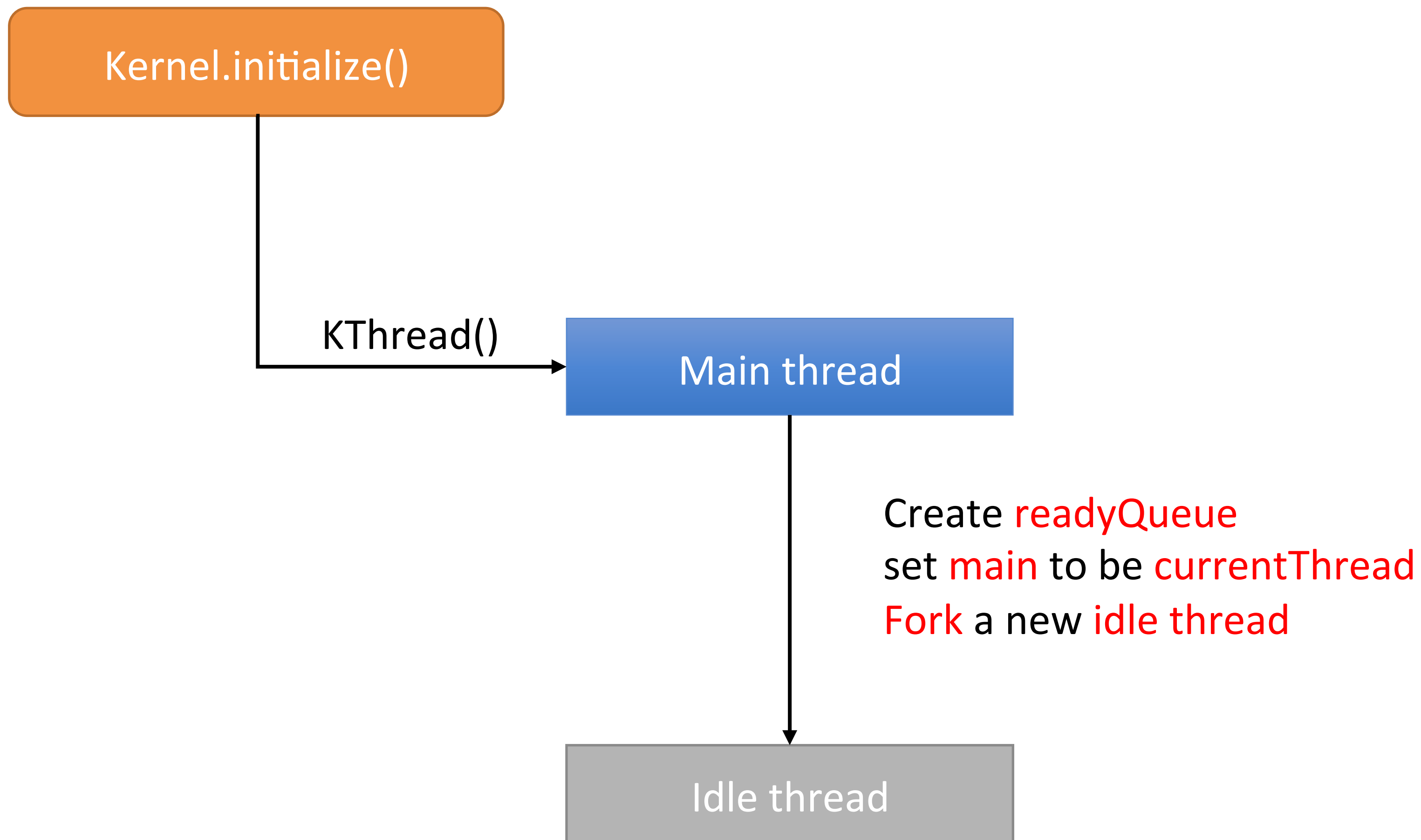
- Thread.restoreState()

Blocked: Sleeping thread waiting for resource

Finished: Thread scheduled for destruction.

- Thread.finish()

From first thread



KThread Methods

KThread.yield()

1. Current thread relinquishes CPU.
2. Add itself to ready queue.
3. Switch to next thread (or idle thread) in ready queue based on scheduler

KThread.sleep()

- Called when current thread has either finished or been blocked.
- If current thread is blocked, then some thread will wake it up.
- Once woken up, it goes back to ready queue for rescheduling.

Project 2 – Part 2 – Task 1

Implement nachos.threads.Condition2

The following methods need to be implemented:

- sleep()
- wake()
- wakeAll()

You may NOT use semaphores.

Project 2 – Part 2 – Task 1

- Structure
  - A queue to store all waiting threads.
- wake()
  - Disable interrupt status
  - Remove first thread in waiting list
  - Put thread in ready queue
  - Restore interrupt status

Project 2 – Part 2 – Task 1

wakeAll()

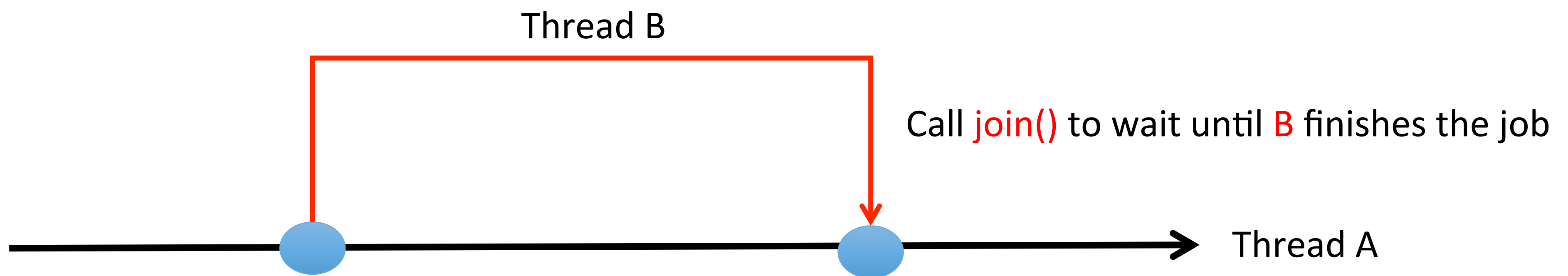
- Disable interrupt status
- Repeat:
  - Remove first thread in waiting list
  - Add thread to ready queue
- Restore interrupt status

Project 2 – Part 2 – Task 1

sleep()

- Disable interrupt status
- Release condition lock
- Add thread to waiting list
- Thread goes to sleep
- Acquire condition lock
- Restore interrupt status

Project 2 – Part 2 – Task 2



- Implement KThread.join()
  - Threads A,B
  - B begins doing a job
  - A waits for B to finish the job (by calling join())

Project 2 – Part 2 – Task 2

Only join once

What if the thread to be joined has terminated?

- Do nothing.

Otherwise wait until the thread finishes, then join it

Project 2 – Part 2 – Task 2

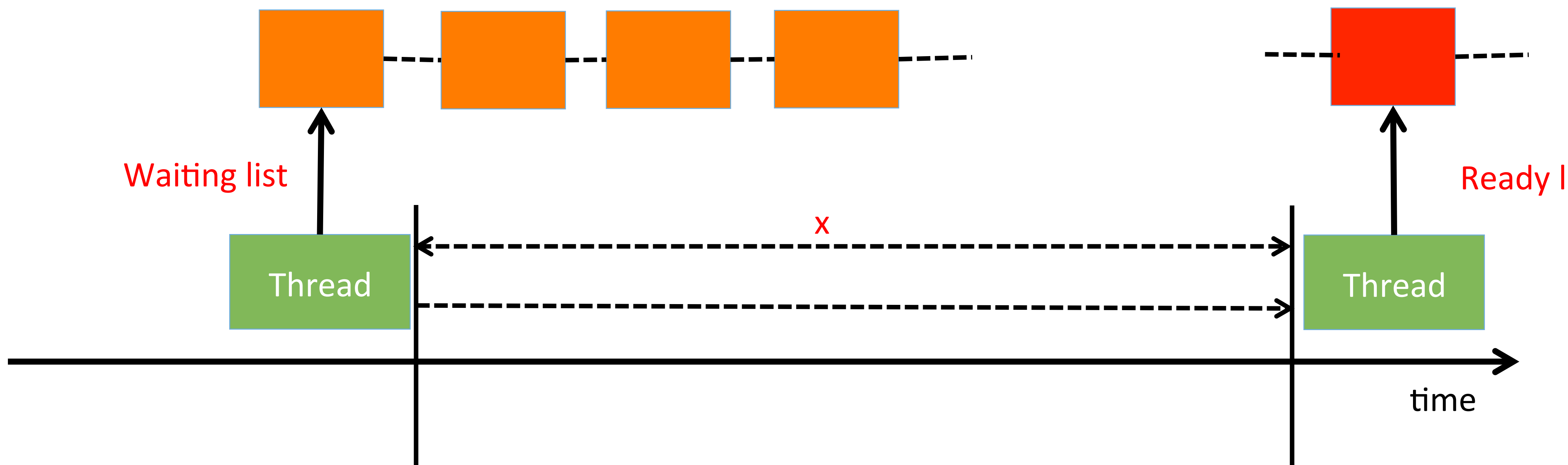
Structure
- A flag to ensure that join() is only used once
- Thread that calls join()

join()
- Check if it can be joined.
- Disable interrupts.
- Terminate the thread.
- Pick up the next thread.
- Restore the interrupt status.

Can only join a child thread
- Should check if called by the parent.

Project 2 – Part 2 – Task 3

- To complete implementation of Alarm class
  - Implement waitUntil(long x)
  - Implement timerInterrupt()



Project 2 – Part 2 – Task 3

waitUntil(long x)

- Disable Interrupt status
- Add thread to the waiting list
  - Use machine.timer.getTime() to get the current time
  - Get_current_time + x
- Put the thread to sleep
- Restore interrupt status

Project 2 – Part 2 – Task 3

timerInterrupt()

- Check if wakeup time is up
- Disable interrupt status
- Remove the thread from waiting list
- Add the thread to the ready list
- Restore interrupt status

Project 2 – Part 2 – Task 4

- Implement Communicator.java
  - speak(int word)
  - listen()
  - Only one lock
  - Do not use semaphores
- Structure
  - Lock: mutual exclusion
  - Condition variable: send message
  - Condition variable: receive message
  - Speaker list
  - Listener list

Project 2 – Part 2 – Task 4

speak(int word)

- Acquire lock
- If the listener list is NOT empty
    - Wake the listener condition
    - Send the word to the first listener and remove the listener.
- Else
    - Add speaker to the speaker list
    - Speaker condition goes to sleep
- Release listener

Project 2 – Part 2 – Task 4

listen()

- Acquire lock
- If the speaker list is NOT empty
  - Wake the speaker condition
  - Receive word and remove the speaker.
- Else
  - Add listener to the listener list.
  - Listener condition goes to sleep.
- Release lock

Testing your implementations

Download test cases from course website.

Put .java files in directory ag

Modify Makefile

- ag = AutoGrader BoatGrader ThreadGrader3 ThreadGrader1 ThreadGrader4 ThreadGrader2

Compile & Run

- Under proj1
- Compile using make
- Run with
  - java nachos.machine.Machine -- nachos.ag.ThreadGrader1
  - java nachos.machine.Machine -- nachos.ag.ThreadGrader2
  - java nachos.machine.Machine -- nachos.ag.ThreadGrader3
  - java nachos.machine.Machine -- nachos.ag.ThreadGrader4
- Modify tester codes to include more test cases.
- Try changing the random seed by including `–s seed` in the command line arguments.

Report

Include “report.pdf” under `nachos/proj1/report.pdf`

In the report, you should include:

- Answers to questions in project description
- Key data structures used/block diagrams of your implementation (2 page limit)

SVN setup (UNIX)

Only ONE group member needs to run these steps.

Checkout the empty repository inside your Nachos directory in the terminal.

- Run `svn checkout` https://.cas.mcmaster.ca/cs3mh3-se3sh3/group#
- Accept any certificates permanently.

Now you should add & commit all your files inside your nachos directory to the SVN.

- `Svn add *`
- `Svn commit *`

SVN setup (UNIX)

Now the rest of the group can run commands using SVN in terminal

```
Svn update which_files_to_update
```

- Update current local working copy of nachos to the latest revision on the repository.

```
Svn commit which_files_to_commit
```

- Commit your changes in your local working copy of nachos to the repository to make a new revision.

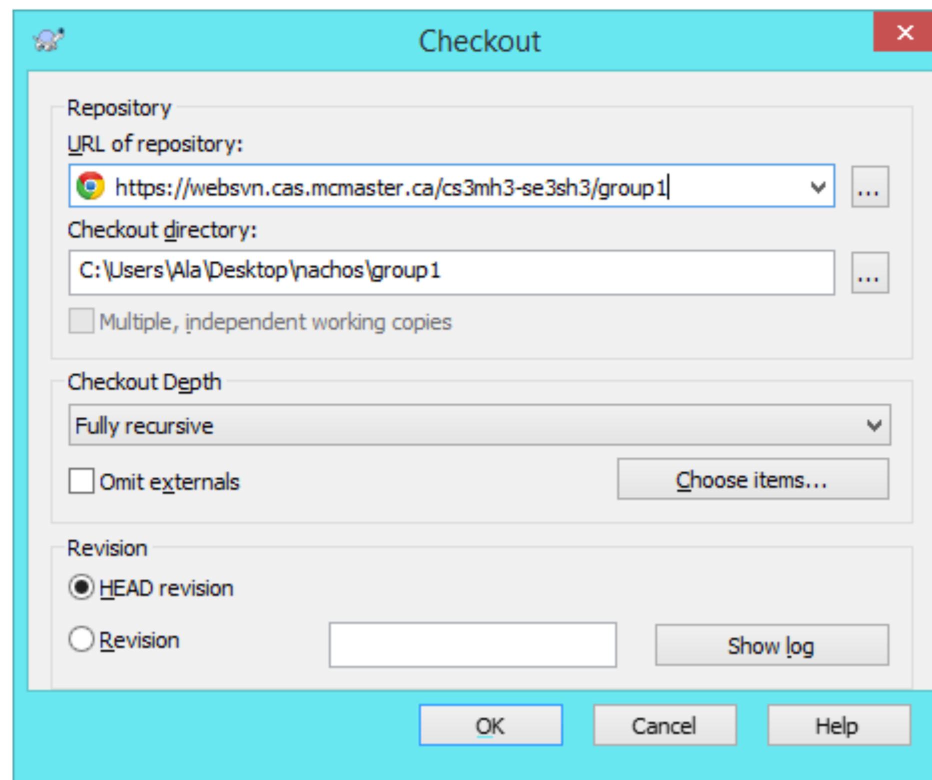
REPOSITORY

Commit

Update

SVN Setup (Windows)

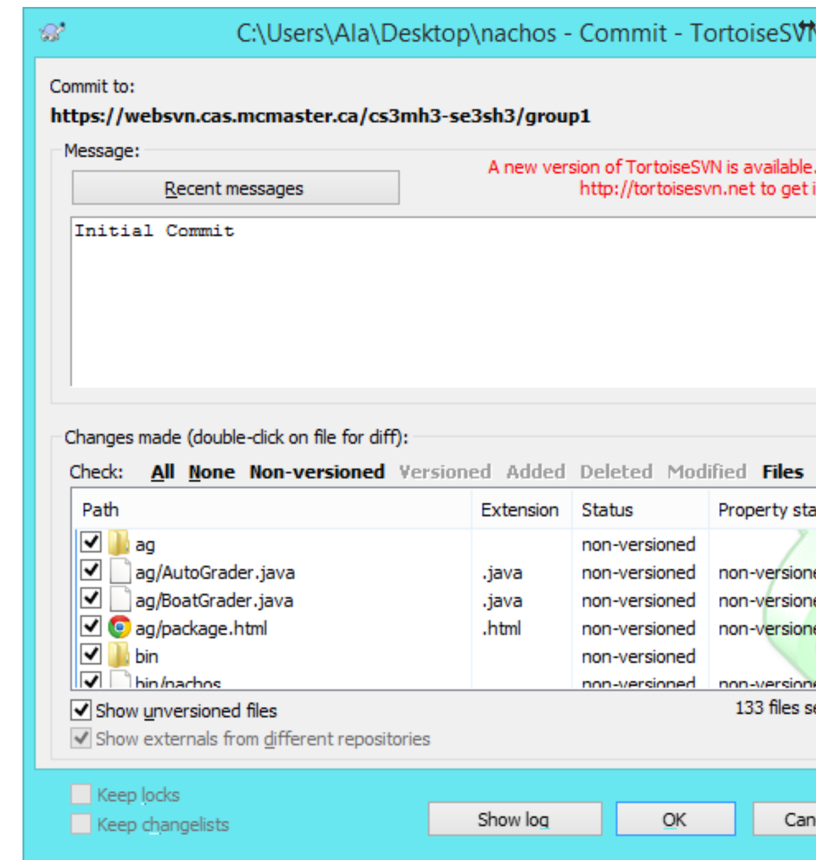
- Download & install TortoiseSVN
- Navigate to your Nachos directory and right click→SVN checkout
- Enter Mac id/pass



SVN Setup (Windows)

- Now you must do an “initial commit”, similarly to UNIX.
- Check all files inside “changes made” box, and add a commit message.
- Now the rest of the group members can update and commit by

Right click->SVN update or SVN commit



Grading

Each project grade is divided as follows:

- 20% Q&A.
- 40% data structures/diagrams.
- 40% implementation of tasks.

NO MSAF is accepted for the projects.

10% late penalty every business day after the due date [Note changes to the percentage].

Start EARLY