## Operating Systems Spring 2015

# Project Instructions

## Required Software Setup

### Nachos Code Distributions

The Java version of Nachos is available for you to download from learn.tsinghua.edu.cn (Under course files filenames are nachos-java.tar.gz. You can unpack this file using the command

tar zxvf nachos-java.tar.gz nachos

which will create the directory nachos. See the file README contained within the tarfile for information on using and compiling the code.

Each Subversion repository comes with the recommended repository structure and the Nachos distribution checked in. You are also free to download the code and look at it without checking it out from SVN. We will be setting up project groups and SVN repositories for each group soon; in the meantime, you should be reading through the code and trying to understand how it works.

You can also view a PDF version of the code and walkthrough (you can download from web learning).

### Java Distributions

The Java version of Nachos requires Java SE Java Development Kit 1.5 or later. If you are not using an instructional machine, you can download the appropriate Java SE JDK for your machine from here: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

### MIPS Cross-compilers

Nachos executes user applications on a simulated MIPS computer. In order to compile these programs, you need a MIPS cross-compiler. One is already installed on the instructional machines; if you are not using an instructional machine, you must download and install the appropriate cross-compiler from learn.tsinghua.edu.cn (Under course files - file name is MIPS.zip)**.**

## General Project Information

There are four projects. Each project has three components:

1. A design document. You will meet with your TA to discuss your design about 7-10 days before your code is due for each project (phase) of Nachos.
2. Project code.
3. Project group member evaluations.

### Project Grading

The design document will be worth 40% of the project grade, and the code itself will be worth the other 60%. You **must** turn in your initial design document in order to get a grade for the project.

### Project Deadlines

You should submit assignments as early as possible to avoid submission problems.

*Assignments (design docs, code, and group evaluations) are due by 11:59pm on the due date.*

### Submission Instructions

Submission instructions for the initial design doc in project 1

* Only one member in the group needs to submit the homework, to minimize prospective confusions, please only designate one member to submit the homework for each group assignment (your group can change the member in charge of homework submission in different assignments). We do deduct points for multiple submissions – for poor team organization and communication.
* During your first deadline, please submit the following files to learn.tsinghua.edu.cn
  + Put your group number and names in the text box
  + Your initial design document, please name it as proj1-initial-design.pdf
* During your second deadline, please submit
  + Place your group number and names in the text box
  + The link to your GitHub repo in the text box
  + Your final design document, please name it as proj1-final-design.pdf
  + Your source code as a tar file (or zip file), generated by the “submit” script, as separately described in each of the four projects.
* You can submit as many times as you want to before the deadline, only the last submission before the deadline counts.

## Design document instructions

### Overview

* + Be sure to read through this entire document. It's all relevant.
  + Each design document is worth 40% of the project. While it will likely take less than 40% of the time you spend on the project, you should take it very seriously.
  + We will grade your designs harshly. The design is essentially the most important part of the project. Having a good project design can literally cut your total coding time by a factor of 10.
  + Design documents should be around 2,000 to 4,000 English words long. If it's longer than 5,000 words, we won't read it. So keep'em short and to the point.
  + Design documents will be submitted online, in PDF or text format. We encourage you to use your favorite word processor to make your design document, but you must convert it to PDF or text when you're done. We'll provide basic directions on how to do that.
  + On learn.tsinghua.edu.cn, we posted a sample project and a sample design document. However, the sample project is only a fraction of the length of your projects.

### What goes into a design document?

A design document is a complete high-level solution to the problem presented. It should be detailed enough that somebody who already understands the problem could go out and code the project without having to make any significant decisions. Further, if this somebody happens to be an experienced coder, they should be able to use the design document to code the solution in a few hours (not necessarily including debugging).

### So what actually goes in?

A high-level description of your solution, including **design decisions** and **data structures**

**Declarations** for all new classes, procedures, and global/class variables

**Descriptions** of all new procedures (unless you can tell exactly what it does from the name), including the purpose of the procedure, and an explanation of how it works and/or pseudocode

For example, this is the pseudocode I would write for the existing Condition::Wait:

void Condition::Wait()

    waiter = new Semaphore, initially 0

    append waiter to waitQueue

    conditionLock->Release()

    waiter->P()

    conditionLock->Acquire()

Your pseudocode has to be precise. For instance, in describing your solution for the Communicator class, it is not enough to say  if anybody's waiting, then signal cv  You need to say how you determine if anybody's waiting, e.g. by saying     if (waitingReceivers>0 or waitingSenders>0) then signal cv

Also, another important thing to remember is that a design document needs to include the **correctness invariants** and **testing strategy**. The testing strategy includes a clear plan of the testing methodology and may include a description of test cases that will be used to test correctness invariants. Focus on the testing strategy. If you want, you may itemize things that will need testing.

### What *doesn't* go into a design document?

Keep in mind that your TA understands the project very well. Do not restate the problem in your design document. Your TA is far more interested in your solution than in knowing that you understood the problem.

## Guide to Nachos

### Nachos Project instructions (first two projects)

The first two projects for this course is to build an operating system from scratch. We believe that the best way for you to understand operating systems concepts is to build a real operating system and then to experiment with it. To help you get started, people have built a very simple, but functional, operating system called Nachos. Over the course of the semester, your job will be to improve the functionality and performance of Nachos; we expect that you will eventually modify or replace much of what we have written.

The project has four phases, corresponding to four of the major pieces of a modern operating system: thread management, multiprogramming, virtual memory, and networking.

Some phases build on previous phases; for example, all parts of the project use thread management routines. As far as possible, we have structured the assignments so that you will be able to finish the project even if all of the pieces are not working, but you will get more out of the project if you use your own software. Part of the charm of building operating systems is that you get to "use what you build": if you do a good job in designing and implementing the early phases of the project, that will simplify your task in building later phases.

Part of the code we provide is a software emulation of a network of MIPS-like workstations. For instance, our code emulates turning on and off interrupts, taking exceptions and page faults, as well as the actions of physical devices (e.g. a disk, a console, and a network controller).

It is important to realize that while we run Nachos on top of this emulation as a Java program, the code you write and most of the code we provide are exactly the same as if Nachos were running on bare hardware. We run Nachos as a Java program for convenience: it is very portable, it is easier to debug (Java is a type-safe language, and we can use jdb), and so that multiple students can run Nachos at the same time on the same machine. These same reasons apply in industry; it is usually a good idea to test out system code in a simulated environment before running it on potentially flaky hardware.

In order to port Nachos to real hardware, we would have to replace our emulation code with a little bit of machine-dependent code and some physical machines. For example, in phase 1, we provide routines to enable and disable interrupts; on real hardware, the CPU provides these functions.

Unless you work for a really smart company, when you develop operating system software you usually cannot change the hardware to make your life easier. Thus, you are not permitted to change any of our emulation code, although you are permitted to change most of the Nachos code that runs on top of the emulation.

Finally, some suggestions for doing well in this course:

Read the code that is handed out with the assignment first, until you pretty much understand it. Start with the interface documentation.

Don't code until you understand what you are doing. Design, design, and design first.

Only then split up what each of you will code.

If you get stuck, talk to as many other people as possible.

### A guide for Nachos

Will be handed out in class. Also posted on learn.tsinghua.edu.cn