Homework 4
Operating Systems, V22.0202
Fall 2007, Professor Yap

Due: Wed Oct 31, 2007.

THE TOY OS PROJECT

This assignment has two independent parts. In PART A, to help you understand THE Machine architecture, we ask you to write a simple STML program. In PART B, you will modify our Toy OS (TOS) which operates on THE Machine. The homework is not technically difficult, but it involves reading and understanding the C code for THE/TOS. We assume that you have read our Class Lecture Notes 5 which describes THE/TOS.

You will work in teams of two. Ideally, I like teams to program in the "extreme programming" (XP) spirit as discussed in class. To program in this style, both of you should sit down at the terminal. Have one person code while the other looks over the shoulders of the first. Exchange roles for different parts of the project. Note that there is a whole philosophy of XP which you can read about on the web; we are taking just a minor part of this.

You will need to use the CVS repository found in `/home/exact/cvsroot`. Details about CVS is found in our webpage `http://cs.nyu.edu/exact/collab/`. Since you have individually (NOT as a team) sent me your public key (id_dsa.pub file), you should be able to access the CVS under the `e-friend` account at `access.cims.nyu.edu`. NOTE: please do not use the `yapclass` account mentioned in the webpage. In particular, your variables `CVS_RSH` and `CVSROOT` must be set as follows

```
% export CVS_RSH=ssh
% export CVSROOT=ext:e-friend@access.cims.nyu.edu:/home/exact/cvsroot
```

EXPLANATION: The value of `CVS_RSH` is the name of the program (i.e., ssh) to use for going to the internet to access the CVS repository. The value of CVSROOT actually has three parts, which separated by colons (":"). The first part "ext" says that the repository is external. This means CVS will use the $CVS_RSH=ssh program to access the repository on the internet. The second part "e-friend@access.cims.nyu.edu" says that you will use the "e-friend" account to log into the internet server "access.cims.nyu.edu". The third part says that the CVS repository is located at "/home/exact/cvsroot".

You use the CVS in two ways: (a) to check out our initial sources, and (b) to deposit your projects. Thus both team members can work independently on the project and still share the same files. Let the variable `BASE` be set to `class/os/2007`. The module containing the initial source code for THE/TOS is `$BASE/stm/stm1` = `class/os/2007/stm/stm1`. Each team will be given a secret TEAMCODE. E.g., if your TEAMCODE is 123456, then you are to store all your work under the module `$BASE/123456` = `$BASE/$TEAMCODE`. In your Makefiles, please use the variables BASE and TEAMCODE.

1 PRELIMINARY

We suggest you prepare for the programming part of the homework by taking these steps. In fact, it is a good idea that each team work together on the following.

- Create a new directory called `hw4` on your computer. Do `cd` to this directory.
- Use CVS to checkout THE/TOS software from the module called `os/stm/stm1`. If you set up the CVS environmental variables properly, as instructed in our webpage, then you can simply issue the command:

```
% cvs co -d myStm1 $BASE/stm/stm1
```

This creates a local copy of the module `stm1` in a new subdirectory called `myStm1` in your computer.

- Next go to the `myStm1` directory, and compile the software (you can simply type `make os`). The Toy OS is stored in an executable file named `tos.exe`.
- In the stm1 module, we provide several sample STML programs. You DO NOT have to figure out how these STML programs work (though they're not difficult). But if you are curious, I would be happy to discuss them with you.

  You can directly use your compiled Toy OS to execute any STML programs. Just type

  ```
  % tos.exe <stml-program>
  ```

  and type any input arguments needed by the stml program. Note that `tos.exe` can take optional arguments (`-d, -m, -b` as discussed below).
- Our Makefile has several targets to run these sample programs. Each will print some helpful hints on usage. Try `make primes` to see the `primes.stml` program in action.
- Since you are going to modify THE/TOS, we next ask you to create a new CVS module called `stm2` which is initially just a copy of `stm1`:

  ```
  % cd ~/hw4
  % cp -r myStm1 tmp --- make a copy of myStm1
  % cd tmp -- go to the tmp subdirectory
  % rm -r CVS --- remove all CVS information
  % --- also remove all temporary files that may be created when you compiled or edited programs
  % --- E.g.  .o files, .exe files, etc.
  % cvs import class/os/2007/TEAMCODE/hw4/stm2 ven rel --- put the contents of tmp back to CVS as
  % --- note that "ven" and "rel" are dummy arguments as far as we are concerned
  % cd ..  --- back to parent directory
  % cvs checkout -d stm2 os/2007/TEAMCODE/hw4/stm2 --- immediately check out stm2
  % rm -r tmp --- assuming checkout was successful, you can remove tmp directory now!
  % cd stm2 --- now you complete your assignment in this directory, and commit changes back to CV
  ```

  Try to understand each of these steps, as you can reuse them for future projects!!
- On the homework page, please take a look at the html file called `examples.html` which contains sample outputs of THE/TOS on the following three runs:

  ```
  % tos -b 1000 -d 2 primes.stm
  ```

  with input 12.

  ```
  % tos -b 50 -d 1 fraction.stm
  ```

  with input $400, 1001, 1000, 20$.

  The last run uses the multi-programming version of the Toy OS, called `tos2`, similar to what you must write:

  ```
  % tos2 -d 2 fraction.stm primes.stm
  ```

  with input 12 to `primes.stm`, and input $400, 1001, 1000, 20$ to `fraction.stm`.

2 PART A: STML PROGRAMMING

Write an STML program to compute the Fibonacci numbers. Call this program `fib.stml`. It takes an input integer n and outputs a sequence of the first n Fibonacci numbers. The Fibonacci numbers are given by the sequence

$$(f_0, f_1, f_2, f_3, \ldots) = (0, 1, 1, 2, 3, 5, 8, \ldots)$$

where $f_0 = 0, f_1 = 1$ and for $i \geq 1$, we have $f_{i+1} = f_{i-1} + f_i$. Note that this sequence grows very fast, so n cannot be too big before f_n exceed 2^{32} and cause an overflow. For instance, if $n = 5$, then your program should print out the sequence $0, 1, 1, 2, 3$.

Naturally, we expect you to test `fib.stml` using the given Toy OS (`tos.exe`) found in `$BASE/stm/stm1`.

After testing, you need to figure out the largest integer n such that your fibonacci program is correct on input n. This value of n must be reported to us in the README file (see below on submission).

Add and commit your STML program into the module `$BASE/$TEAMCODE/hw4`.

3 PART B: MULTI-PROGRAMMING

Our original Toy OS runs one process at a time. We want you to modify the Toy OS so that it supports multiple processes. The actual programming involves writing about 80 lines of straightforward C code; in a more compressed style, you may be able to get away with 40 lines. It requires no difficult analysis, only simple features of C, and no clever systems hackery.

Work through the C code provided, and figure out how the relevant parts of the code work. The internal workings of the procedures `load_stm`, `show_inst`, `get_inst`, and `exec_inst` do not actually matter to anything you will be doing – you only have to understand what they do.

You are to produce two files, `tos2.c` and `tos2.h`. These are compile and link to the original `the.h, the.c`. You must not modify `the.h, the.c`.

Here is how tos2 is supposed to work: suppose you want to execute three processes where the first process runs `primes.stm` and the second runs `gcd.stm` and the third runs `primes.stm`. You type:

```
% tos2 [-d nnn] [-m nnn] [-b nnn] [-q nnn] primes.stm gcd.stm prime.stm
```

Here we use the convention that any argument inside `[...]` is optional. The flag `-d` specifies the debugging level, the flag `-m` specifies the maximum number of instructions, and the flag `-b` specifies the base address for loading the program. The values `nnn` are integer arguments for these flags (each occurence of `nnn` may be different). If these arguments are not given, default values will be used. Moreover, you can give these values for each of the processes to be run. E.g.,

```
% tos2 -d 1 primes.stm -d 2 -m 20 gcd.stm -m 100 -d 1 prime.stm
```

But the flag `-q` is new, which you need to implement: it tells the OS what time quantum to use for each processes. We assume a simple round robin scheduling.

REMARK: there is an issue that the current assignment will not attempt to solve. Namely, the input/output of the two processes can get intermixed. Hopefully, it will not be too confusing for the current project. But there are two ways to run these programs so that the outputs from these processes can still make sense. One is to set the quantum (using flag `-q`) to a very large number. This effectively forces sequential execution. The second way is to use debugging level 1 to get some useful prompts during I/O traps (see the source code for what gets printed).

To solve these issues, we can introduce semaphores. This will be addressed in the future assignment.

Submitting your work.

1. Since all your work is in CVS, there is nothing to submit explicitly! But whatever we find in the module `$BASE/$TEAMCODE/hw4` will constitute your submission.

   After midnite of the due date, you must stop working on this module (NOTE: that we can check the time-stamp on each file and directory to see when you last worked on a file or directory).

2. All the RELEVANT instructions from previous programming assignments still apply. In particular, you must have a Makefile and a README file in each directory of the repository.

3. For this assignment, we also expect to see the files "fib.stm", "tos2.h" and "tos2.c", as well as the original files "tos.h, tos.c, the.h, the.c" and the sample STML files.

4. Make sure that all your programs are properly commented, and easy to read (use proper indentation, etc).

5. The README file must identify this project and your team, and it MUST give an overview of your modifications. Data structures and important routines must be identified and explained here. We will be reading your programs while guided by your README file.

6. The Makefile should be an adaptation of the original Makefile, and should include these targets:

   - For PART A, we want to see a target called "fib" that compiles "tos" (original) and execute your "fib.stm". In the README file, tell us the first value of n such that fib.stm will fail on input n.
   - PART B: the DEFAULT target (i.e., the first one) should simply compile "tos2" AND run tos2 on some interesting set of inputs. (Note that the original "tos" target should remain there, as we use it in PART A.)
   - Target "tos2" should just compile tos2.c.
   - Note that targets "t1", "t2", "t3", "t4" are already in our Makefile. You should create another target called "t5" that runs "tos2" on other interesting examples. (Try to use "echo" is helpful ways as in our examples.)

Grading.

- This homework is worth 120 points.
- Following our submission instructions is worth up to 20 points (i.e., Makefile, README, etc).
- PART A is worth 20 points: for the "fib.stm" code and its Makefile target.
- The correctness of "tos2" is worth 50 points.
- Good program structure and adequate commenting of "tos2" sources are worth up to 20 points.
- Your "tos2" program will be tested on examples other than those provided here. Worth 10 points.
- If "tos2" does not compile, you get a maximum of 30 points for PART B.