# Project 1A COM S 352 Spring 2023

# (Due: Thursday, Feb 9, 11:59pm)

This project is to be completed by yourself. You may ask classmates for help installing software, setting up your development environment, and understanding the sample program.

## 1. Preparation

As a pre-requisite of this project, you should have set up the xv6-riscv system. Refer to the document titled "Introduction to xv6 (part I)" on Canvas for more instructions on system installation. In addition, if you prefer to install the system on your own native/virtual Linux/macOS machine, refer to https://pdos.csail.mit.edu/6.828/2019/tools.html.

## 2. Test and read an example

Attached is a sample program, name "typist.c", which counts the number of characters a user types.

### 2.1 Add the program to your xv6-riscv system

Download the program file "typist.c" and save it to the directory of xv6-riscv/user. Then, add "_typist" to the end of UPROGS in Makefile in directory xv6-riscv/. That is, UPROGS in Makefile should be as follows:

```
…
UPROGS = \
        … …
        $U/_zombine\
    $U/_typist\
...
```

### 2.2 Run the program in the shell

Type "make qemu" to compile the programs and runs the shell.

In the shell, type "typist" to start the program.

With this program, you type texts line by line, where each line should be less than 128 characters. Each line is submitted when the <RETURN/ENTER> key is clicked. The program counts the number of characters you submit for each minute.

### 2.3 Read the program

Read the program to understand how it implements the above functionalities.

In the main function, system call “pipe” is made to have two pipes created. Then, system call “fork” is made to create a child process. Now, we have two processes (parent and child) run concurrently, and the processes share two pipes for communication.

The child process runs in a loop. At each iteration, it calls function “gets” to capture user’s input which is one line of characters, and writes (by making system call “write”) the line to the first pipe (identified by the file descriptors in fd1). If the input line starts with “:exit”, which indicates that the user wants to quit, an arbitrary character (it is ‘a’ in this program) is written to the second pipe (identified by the file descriptors in fd2) and the child process exits; otherwise, the child process goes on with the next iteration.

Concurrently, the parent process also runs in a loop. At each iteration, it first sleeps for one minute (60 seconds). Then it makes system call “read” to pull all characters from the first pipe, gets the number of such characters, and prints out a message stating how many characters were entered in the last minute. It also pulls out all content of the second pipe. If the pipe has content from the child process, meaning the child wants it to exit, it waits for the child process to terminate and then exits; otherwise, it goes on with the next iteration.

Pay attention to how the system calls are used in the program.

## 3. Develop a robot typist program

Now you are required to develop a new program we will call “robottypist.c”, based on revising program “typist.c”.

Program “robottypist.c” should take two arguments, the running time of the robot (in seconds) and the time interval (in seconds) for the robot to type (i.e., generate input). Let’s denote the first argument as A and the second as B; A should be a multiple of B.

Program "robottypist.c” should work in the similar way as “typist.c” but differ in the following: The child process will not wait for typing from the user, but instead automatically generate

input line “Hello!” once every B seconds. Also, after A seconds have passed, the child process should **exit** and inform the parent process to exit.

You are free to copy the code from “typist.c” when develop “robottypist.c”. You should add “_robottypist” to Makefile, so that it can be run in the shell after the new system is compiled.

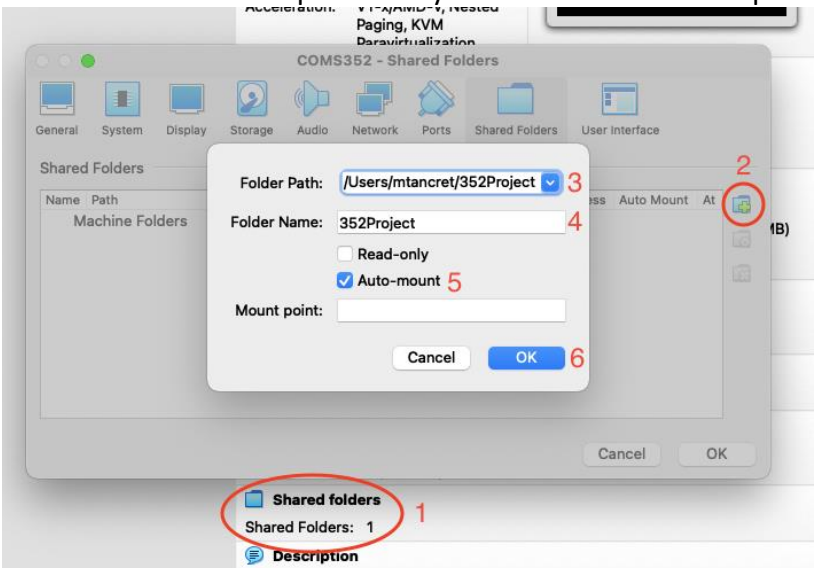
## 4. Submission

You are required to submit two files: “robottypist.c” and Makefile. Add comments to the c program to help grader understand your code. Make sure the program compiles without any errors.

## 5. Notes: How to Copy Files out of the Virtual Machine

These instructions are assuming the use of VirtualBox, other virtualization software have a similar way to share files.

The simplest way to move files from the virtual machine to your host machine is by creating a shared folder. Shutdown the virtual machine if it is currently running, and on the main Manager window click on Shared Folders. Navigate to a Folder Path on your where you want to store project files, select auto-mount but keep read-only unselected and mount point empty.



After restarting the virtual machine you will be able to copy files into the directory `/media/sf_folder_name_you_chose`. Those files will be available at the folder path you chose on your host machine.