多處理機平行程式設計 2021 fall 作業一說明

Problems

> 50 points for each problem. Using tree-structured communication can get up to 50 points. Using serial communication can only get up to 40 points.

1. Circuit Satisfiability Problem

The Circuit Satisfiability Problem (https://en.wikipedia.org/wiki/Circuit_satisfiability_problem) for a given binary circuit is to find the set of inputs that cause that circuit to produce `1` as its output.

Given the C program (can be downloaded from moodle):

```
/* circuitSatifiability.c solves the Circuit Satisfiability
 *  Problem using a brute-force sequential solution.
 *
 *   The particular circuit being tested is "wired" into the
 *   logic of function 'checkCircuit'. All combinations of
 *   inputs that satisfy the circuit are printed.
 *
 *   16-bit version by Michael J. Quinn, Sept 2002.
 *   Extended to 32 bits by Joel C. Adams, Sept 2013.
 */

#include <stdio.h>     // printf()
#include <limits.h>    // UINT_MAX

int checkCircuit (int, int);

int main (int argc, char *argv[]) {
   int i;               /* loop variable (32 bits) */
   int id = 0;           /* process id */
   int count = 0;        /* number of solutions */

   for (i = 0; i <= USHRT_MAX; i++) {
      count += checkCircuit (id, i);
   }

   printf ("Process %d finished.\n", id);
   fflush (stdout);

   printf("\nA total of %d solutions were found.\n\n", count);
   return 0;
}

/* EXTRACT_BIT is a macro that extracts the ith bit of number n.
 *
 * parameters: n, a number;
 *             i, the position of the bit we want to know.
 *
 * return: 1 if 'i'th bit of 'n' is 1; 0 otherwise
 */

#define EXTRACT_BIT(n,i) ( (n & (1<<i) ) ? 1 : 0)


/* checkCircuit() checks the circuit for a given input.
 * parameters: id, the id of the process checking;
 *             bits, the (long) rep. of the input being checked.
 *
 * output: the binary rep. of bits if the circuit outputs 1
 * return: 1 if the circuit outputs 1; 0 otherwise.
 */

#define SIZE 16
```

```
int checkCircuit (int id, int bits) {
   int v[SIZE];        /* Each element is a bit of bits */
   int i;

   for (i = 0; i < SIZE; i++)
     v[i] = EXTRACT_BIT(bits,i);

   if (  (v[0] || v[1]) && (!v[1] || !v[3]) && (v[2] || v[3])
       && (!v[3] || !v[4]) && (v[4] || !v[5])
       && (v[5] || !v[6]) && (v[5] || v[6])
       && (v[6] || !v[15]) && (v[7] || !v[8])
       && (!v[7] || !v[13]) && (v[8] || v[9])
       && (v[8] || !v[9]) && (!v[9] || !v[10])
       && (v[9] || v[11]) && (v[10] || v[11])
       && (v[12] || v[13]) && (v[13] || !v[14])
       && (v[14] || v[15])  )
   {
      printf ("%d) %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d \n", id,
         v[15],v[14],v[13],v[12],
         v[11],v[10],v[9],v[8],v[7],v[6],v[5],v[4],v[3],v[2],v[1],v[0]);
      fflush (stdout);
      return 1;
   } else {
      return 0;
   }
}
```

Using your favorite text editor, modify `circuitSatisfiability.c` so that it uses the `MPI_Wtime()` function to time the computation:

```
double startTime = 0.0, totalTime = 0.0;
startTime = MPI_Wtime();

for (i = 0; i < USHRT_MAX; i++) {
    count += checkCircuit(id, i);
}

totalTime = MPI_Wtime() - startTime;
printf("Process %d finished in time %f secs.\n", id, totalTime);
```

With these modifications, the program will self-report how long it took to check the circuit.

Use MPI’s point-to-point communications to sum the distributed processes’ count-values into a global count, and have process 0 output this global count.

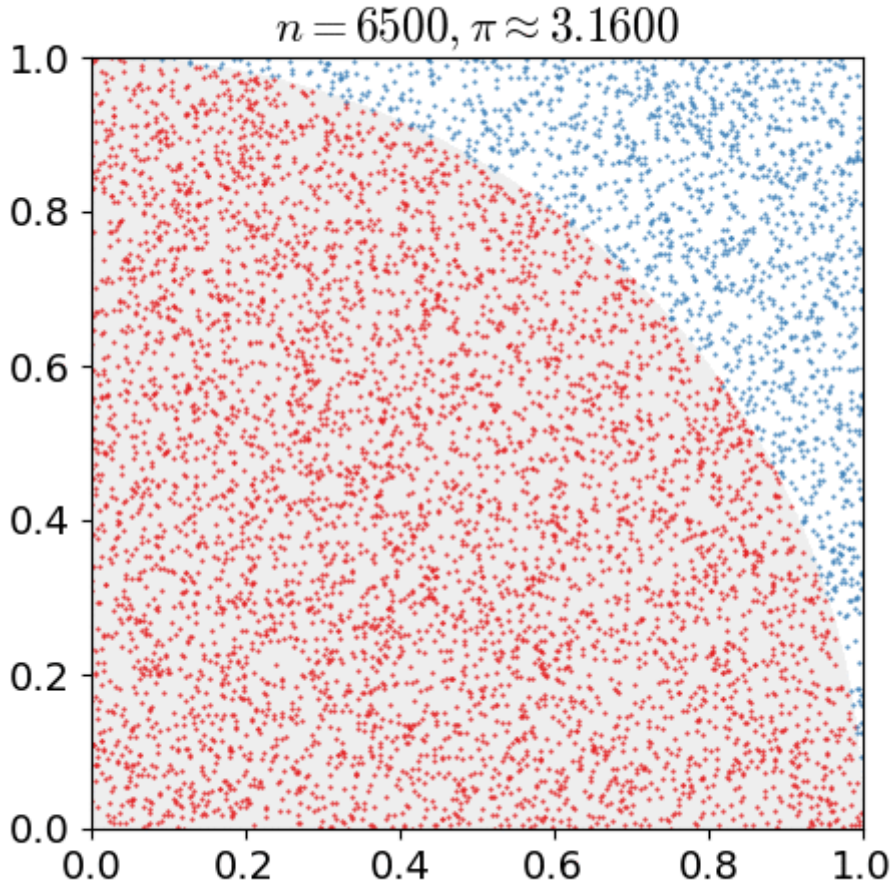
2. Monte Carlo method

Suppose we toss darts randomly at a square dartboard, whose bullseye is at the origin, and whose sides are 2 feet in length. Suppose also that there's a circle inscribed in the square dartboard. The radius of the circle is 1 foot, and it's area is π square feet. If the points that are hit by the darts are uniformly distributed (and we always hit the square), then the number of darts that hit inside the circle should approximately satisfy the equation

$$\frac{number\ in\ circle}{total\ number\ of\ tosses} = \frac{\pi}{4}$$

Since the ratio of the area of the circle to the area of the square is $\pi/4$. We can use this formula to estimate the value of π with a random number generator. (More on Monte Carlo method (https://en.wikipedia.org/wiki/Monte_Carlo_method))

```
number_in_circle = 0
for (toss = 0; toss < number_of_tosses; toss++){
    x = random double between -1 and 1;
    y = random double between -1 and 1;
    distance_squared = x*x + y*y;
    if (distance_squared <= 1) number_in_circle++;
}
pi_estimate = 4 * number_in_circle/((double)number_of_tosses)
```



▲ An example of using Monte Carlo method.

By nicoguaro (https://commons.wikimedia.org/wiki/User:Nicoguaro),CC BY 3.0

Write an MPI program that uses a Monte Carlo method to estimate π. Process 0 should read in the total number of tosses and broadcast it to the other processes by using point-to-point communications. Use point-to-point communications to find the global sum of the local variable number_in_circle, and have process 0 print the result. You may want to use `long long int` for the number of hits in the circle and the number of tosses, since both may have to be very large to get a reasonable estimate of π.

~~(Optional) How to use docker to test your MPI program locally~~

~~We will teach you how to use virtual machines to self-host cluster later in the course. TAs assume you have basic knowledge on docker in this section. If you don't, you still can challenge yourself. What is docker? (https://en.wikipedia.org/wiki/Docker_(software))~~

> 👷 ~~You can help expand this section into a more informative guidelines for beginner~~

- ~~Install docker (https://docs.docker.com/engine/install/) and docker compose (https://docs.docker.com/compose/install/)~~
- ~~Add your username into `docker` usergroup~~
- ~~clone https://github.com/oweidner/docker.openmpi (https://github.com/oweidner/docker.openmpi)~~
- ~~Follow the usage in `README.md`~~

~~Quick usage fix for docker compose V2: `$ docker compose up --scale mpi_head=1 --no-recreate mpi_head --scale mpi_node=3 --no-recreate mpi_node -d`. You may need this since some distribution like ArchLinux has updated the package to V2.~~

Haven't fully tested. (you can still try though)

How to use real cluster to test your MPI program

We have prepared a 3-computer cluster (originally 4, 1 dead recently. Servers donations are welcome) to run your MPI program.

Specification
Core i7 2.93GHz * 4: pn1, pn2, pn3(dead), pn4

> Windows version below ⏬

1. ssh login to 140.116.154.66 at port 22
    - `ssh username@140.116.154.66`

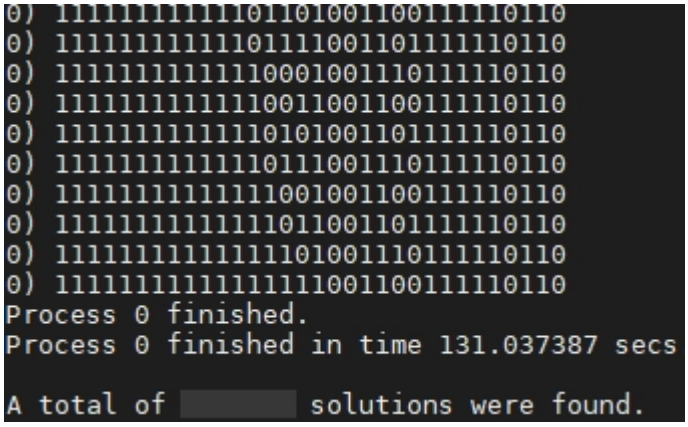
- substitute `username` with your student ID. First character needs to be capital.
- Default password is your student ID. Use `passwd` to change your password after login.

2. Remove `pn3` from `~/mpd.hosts` (Because `pn3` is currently dead)
   - There is a `mpd.hosts` file under your home direct. (You can check it with `ls`)
   - `vim ~/mpd.hosts` . Use down arrow key or `j` move to the line containing `pn3` . Double press `d` to delete that line. Type `:wq` to save and quit.
3. After logging into `pn1` you can `ssh pn2` `ssh pn4` to connect to other servers. `exit` to close the connection to other servers. You will find that you need to type password everytime you `ssh` to other servers. Normally you don't need to connect to other servers. All the works can be done on pn1 since we have mounted disks as NFS (https://en.wikipedia.org/wiki/Network_File_System).
4. Execute the shell script to generate ssh key pairs for yourself to login without password. MPI needs password-less login to switch between machines.
   - `sh ./sshwithout.sh` . Keep pressing enter if it asks you to fill something.
   - (Optional) You can generate another key pair on your computer to log into pn1 without password as well. (More on how to use ssh and how it works (https://wiki.archlinux.org/title/SSH_keys#Copying_the_public_key_to_the_remote_server))
   - `ssh pn2` and `ssh pn4`
   - Type `yes` if it asks you to continue connecting (this will add the host to known host list)
5. Start MPI cluster
   - `mpdboot -n 3 -r ssh`
   - `mpdtrace` to check which servers have MPI booted
   - If Cluster reboots, you will need to `mpdboot` again. TAs will notify you when it happens.

6. Edit/Upload code
   - **Edit:** Servers got `vi` and `vim` installed. You can use them to edit your code (How to use (https://vim.rtorr.com/)/setup (https://hackmd.io/@sysprog/HJv9naEwl?type=view#Vim-%E8%A8%AD%E5%AE%9A) vim). Most terminals use `ctrl + shift + v` hotkey to paste. Or right click to open context menu to paste.
   - **Upload:** Use sftp (https://man.archlinux.org/man/sftp.1.en) to upload your code. You may want to use sftp client like FileZilla (https://filezilla-project.org/) (or other file manager with sftp client built-in, e.g. nautilus, the built-in file manager of Gnome) to upload your file if you are not familiar with sftp.
7. Compile your program

- `cd` to the directory of your code
- `mpiicc -o filename.out filename.c`
- Substitute `filename` with your file name. You may use any name for your file but please use `problem1.c` and `problem2.c` for your final version. TAs will only judge `problem1.c` and `problem2.c`.

8. Run your MPI program
   - `mpiexec -n 8 ./filename.out`
   - n is the amount of cpus to be used
   - 4 cores (hyperthreaded) on each node, 12 total. You can test with different core counts to see how execution time changed.



▲ An example of a 32-bit circuit checker. (You only need to implement the given 16-bit version)

⚠️ Please note that server resources are limited. Don't test your program at the last moment (potentially altogether). Also TAs will try to kill processes if servers are overloaded.

SSH on Windows

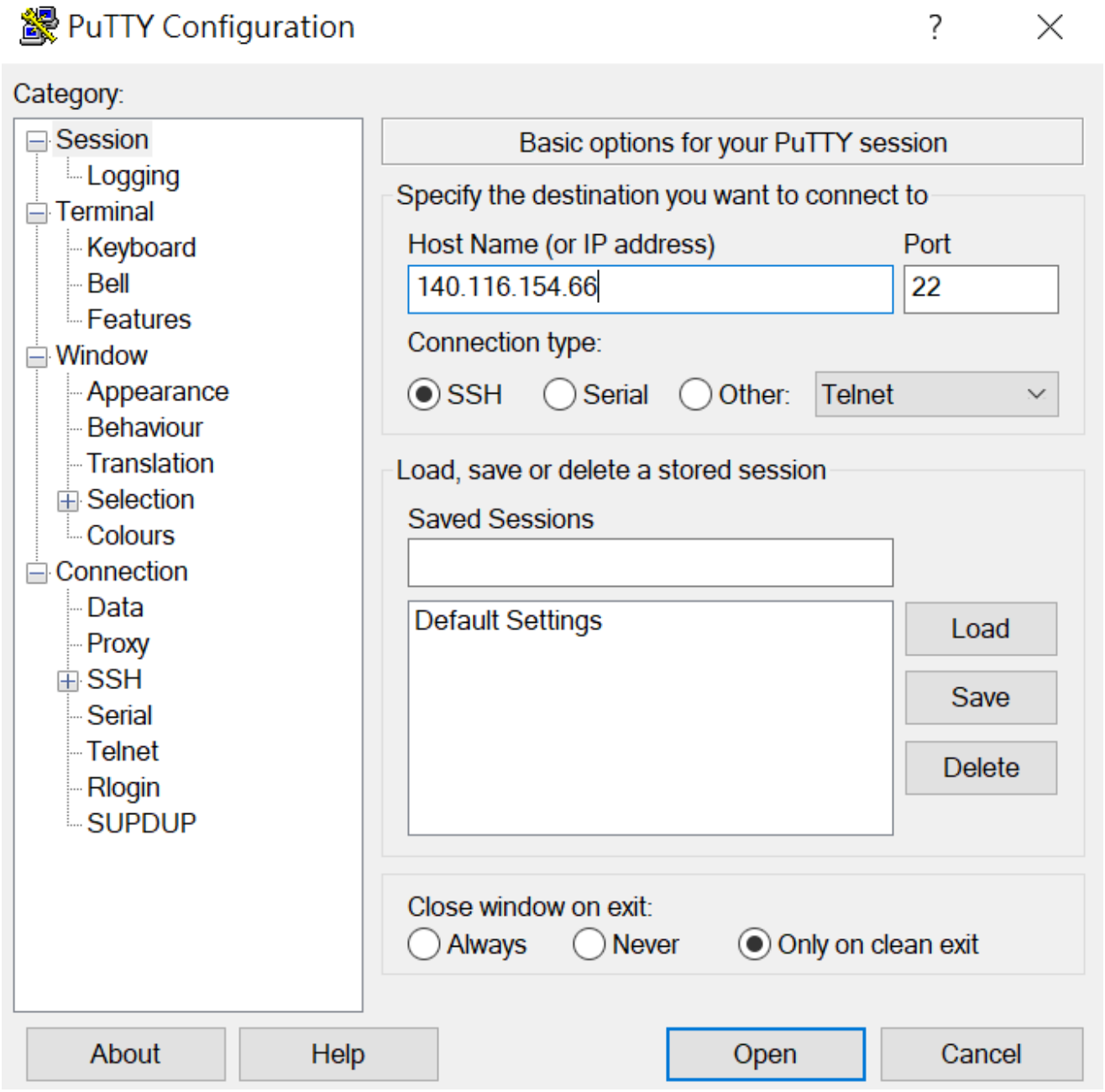
For Windows users, please download **Putty (https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)** or **MobaXterm (https://download.mobatek.net/2132021082033134/MobaXterm_Portable_v21.3.zip)**

Follow steps below to login to the cluster :

Putty

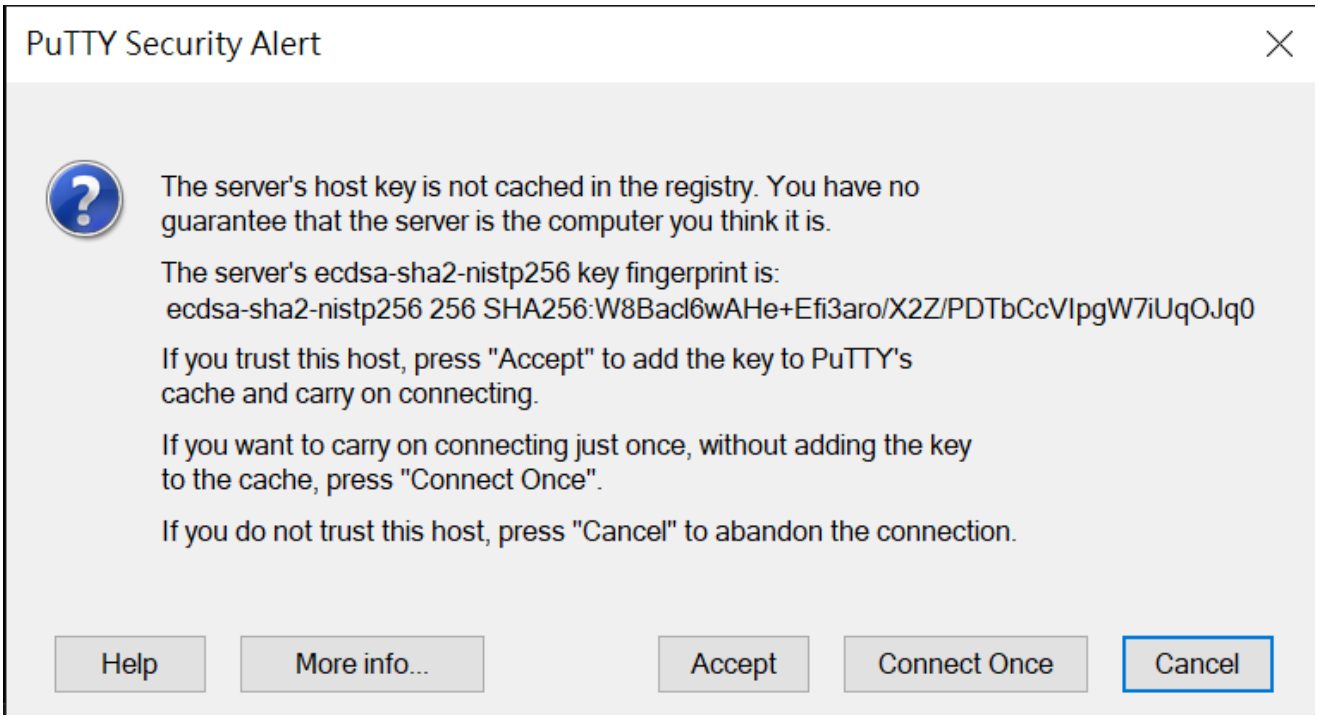
1. Download Putty binary file (https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)
2. Double Click on the Putty binary file

3. Type in the IP address (140.116.154.66) and set port to 22, make sure **SSH** is checked.

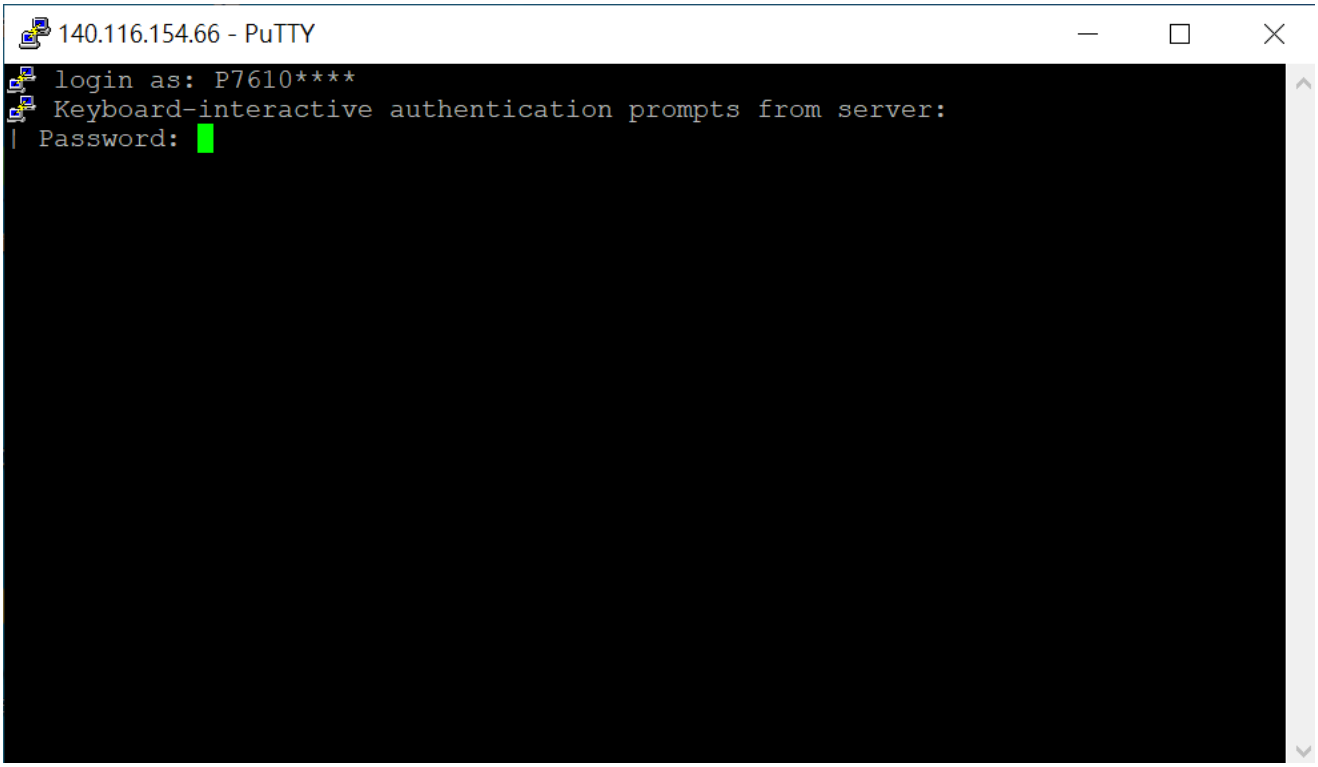


4. Click `Open`

5. You will see a screen like the image down below, Click Accept.



6. Type in your username and password , which is your **student ID**. First character needs to be capital.



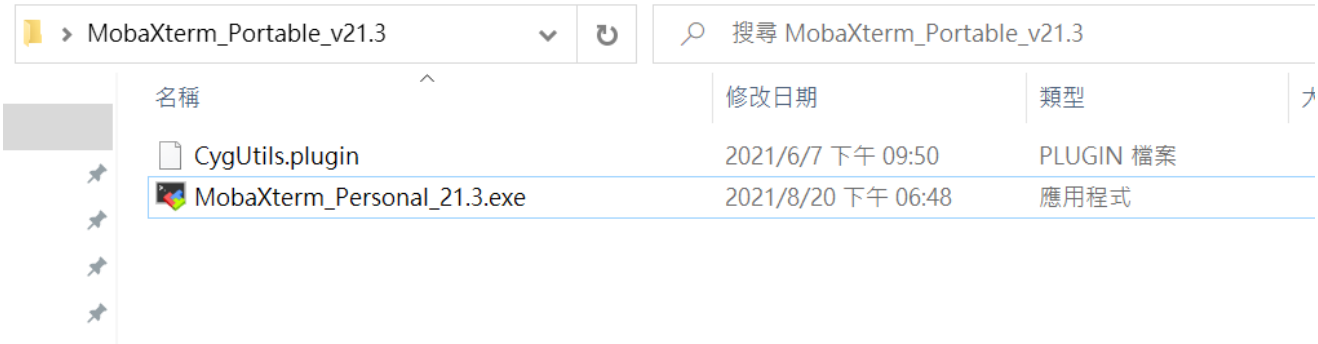
   - Default password is your student ID. Use `passwd` to change your password after login.
   - It is normal that the password you typed in is not visible. Don't be nervous. Your keyboard is not broken. 😄

MobaXterm

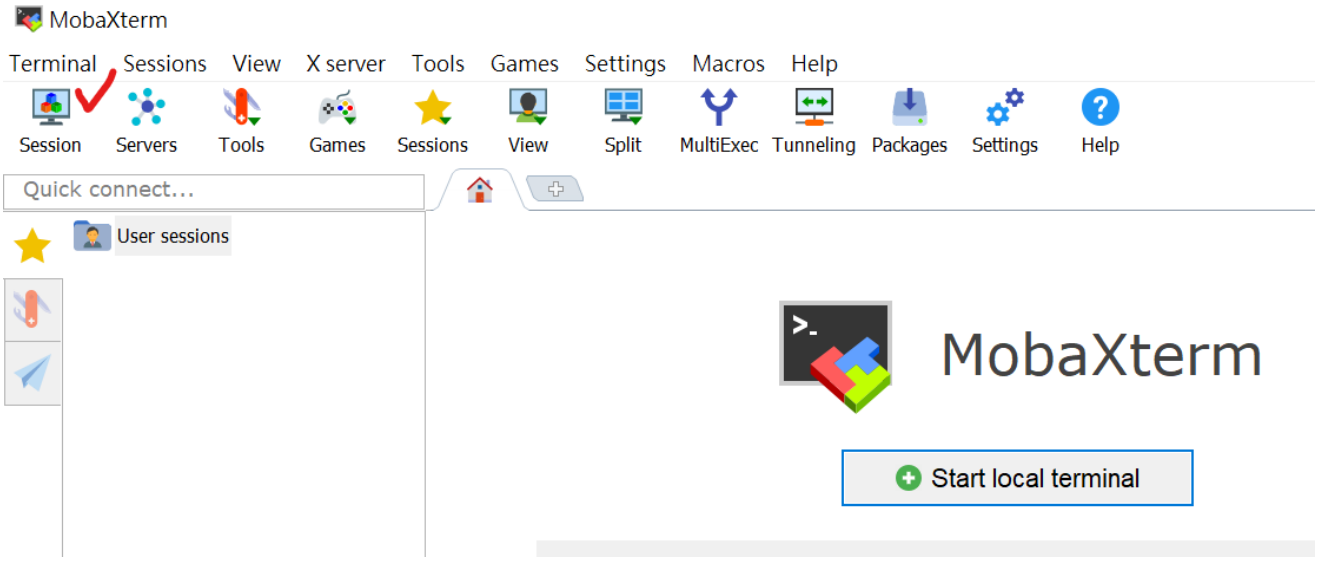
1. Download Portable MobaXterm

   (https://download.mobatek.net/2132021082033134/MobaXterm_Portable_v21.3.zip)

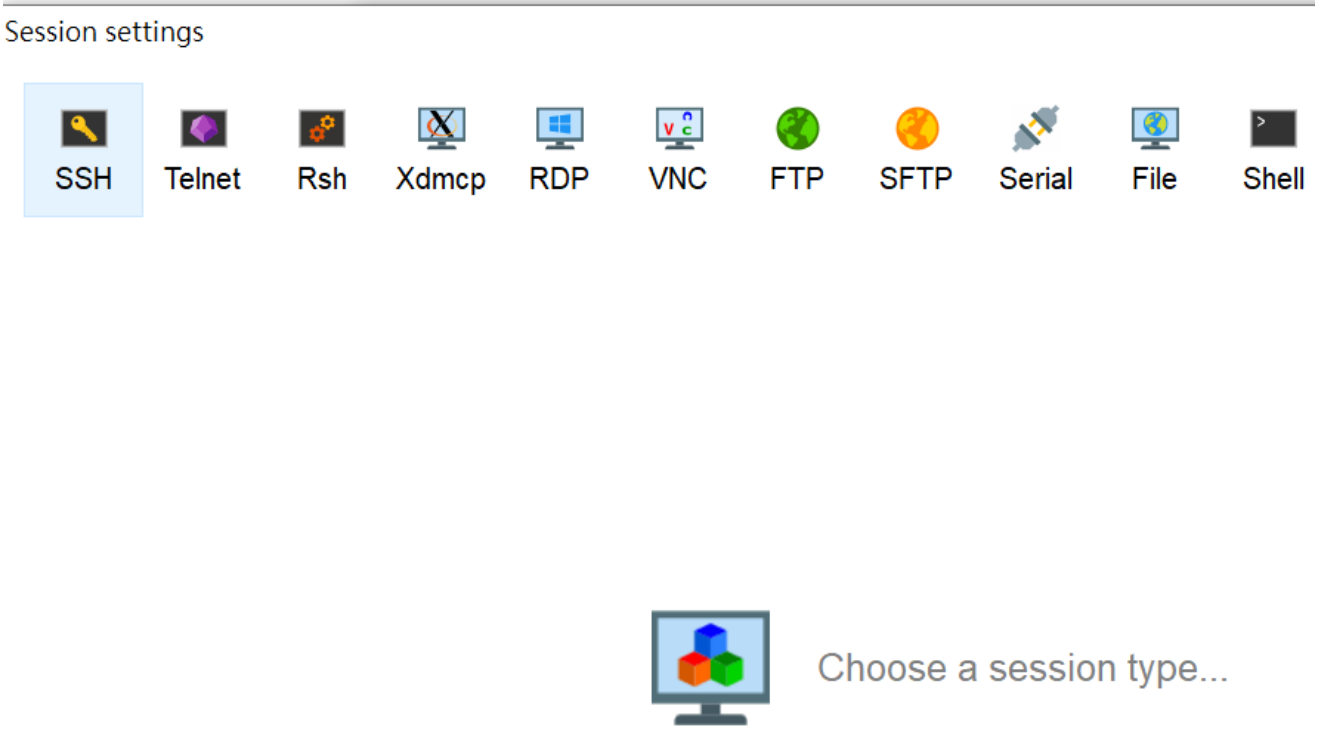
2. Unzip the Portable zip file
3. Double click on the MobaXterm exe



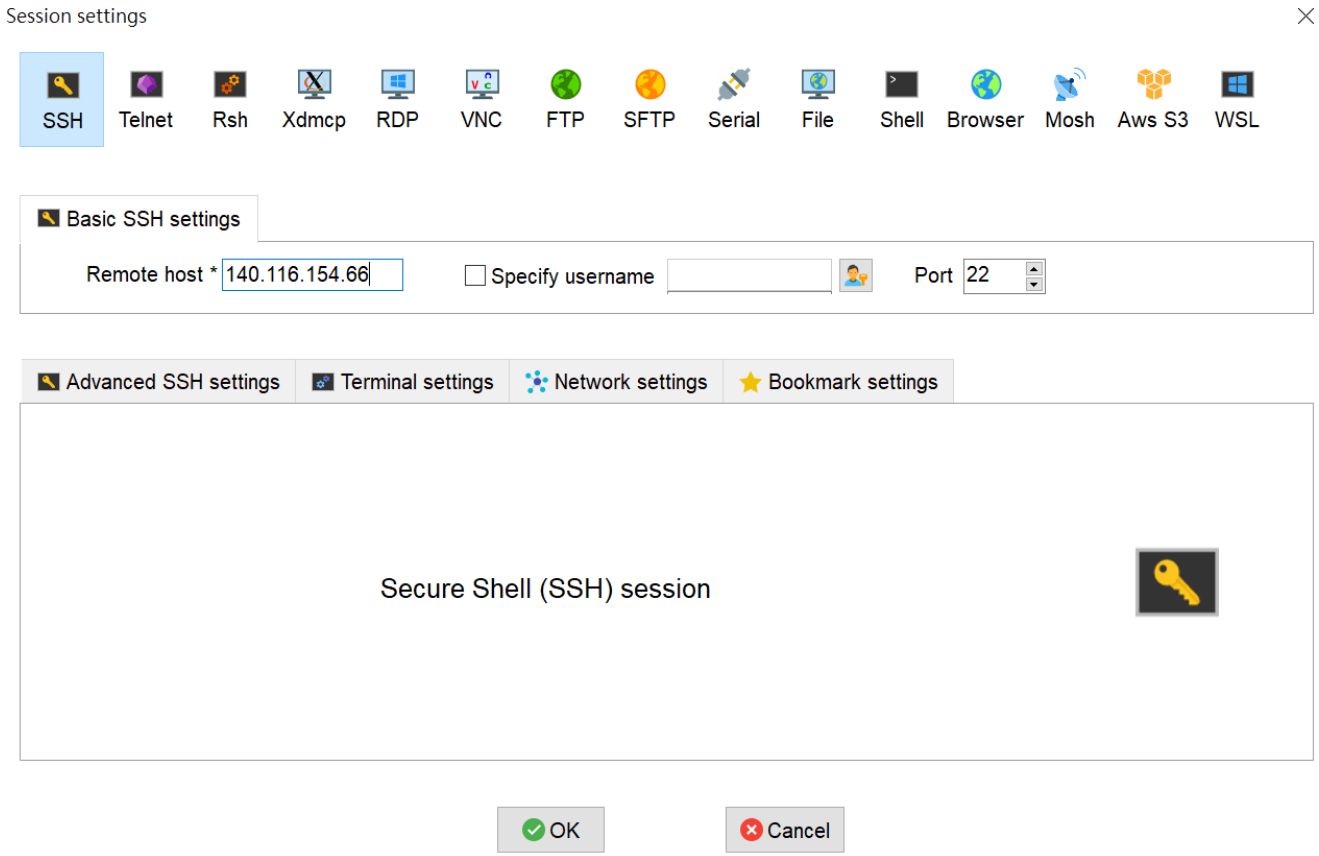
4. Click on Session to create new session



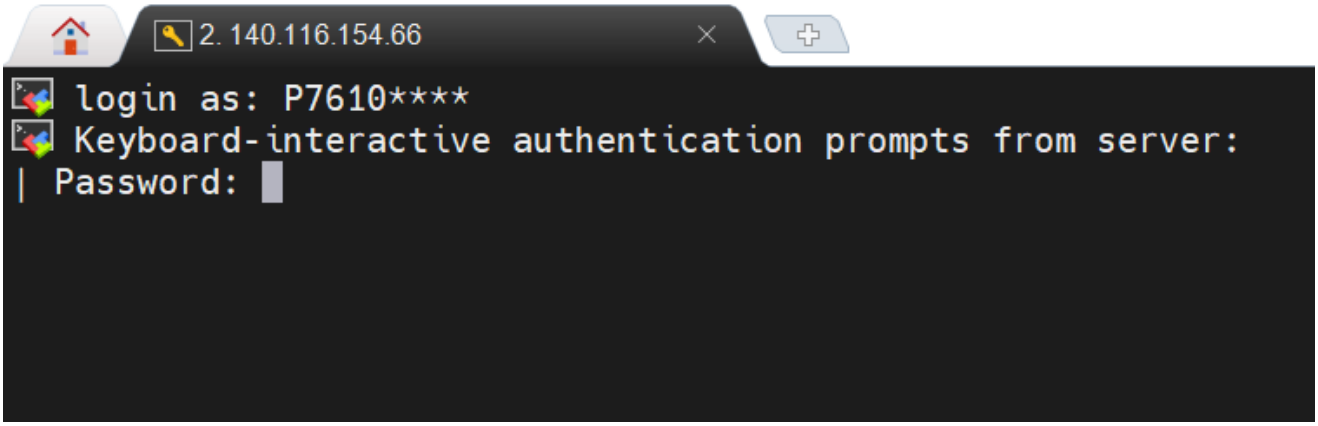
5. Click on SSH



6. Type in the IP address (140.116.154.66) and set port to 22



7. Click `OK`
8. Type in your username and password , which is your **student ID**. First character needs to be capital.



   - Default password is your student ID. Use `passwd` to change your password after login.
   - It is normal that the password you typed in is not visible. Don't be nervous. Your keyboard is not broken. 😄

How to hand in the homework

1. Leave your code under your home directory (`mv yourfile.c ~`) with correct file name: `problem1.c` for problem 1 and `problem2.c` for problem 2. Please don't copy others' code.
2. Upload a report (in `.md` or `.pdf`)(in Chinese or English) to moodle about

- What have you done
- Analysis on your result
- Any difficulites?
- (optional) Feedback to TAs

Deadline: 2021/10/15 23:59:59

Please report any server mis-configuration you found. TAs are new to System/Network administration. We will appreciate your report.

Grading policy

50 points for each problem. Using tree-structured communication can get up to 50 points. Using serial communication can only get up to 40 points.

- Style 25%
  - Not limited but including indentation, comments, readability…etc.
- Correctness 20%
  - Paralleled program should have the same result as serial program.
- Efficiency 30%
  - Get points by making your paralleled program faster than serial program
  - Points are only counted when you have correct result.
- Report 25%
  - We would love to see interesting reports.

Useful links

- https://www.open-mpi.org/doc/ (https://www.open-mpi.org/doc/)
- https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top.html (https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top.html)
- Course forum on moodle
  - You can post your questions there.
  - If you can help answering the questions, TA will try to add more points to you.