香港城市大學
City University of Hong Kong

Introduction to C Language Programming

CS3103 Operating Systems

專業 創新 胸懷全球
Professional · Creative
For The World

Content

- C Basics
  - Read/write data from the standard input/output
  - Command line arguments
  - Pthread library
  - Allocate/release memory
- Environment & Evaluation
  - IDE
  - Compile the C program
  - Evaluate by the Linux redirection

CityU

C Basics

- Read/write data from the standard input/output
- Command line arguments
- Pthread library
- Allocate/release memory

Read/write data from the standard input/output

- In C programming, the following function reads/writes formatted data from standard input/output.

int scanf(const char *format, …);

int printf(const char *format, …);

```
#include <stdio.h>
int main() {
    // Read two integers from stdin
    int n, m;
    scanf(“%d%d”, &n, &m);
    // Write two integers separated by the tab character to stdout
    printf(“%d\t%d\n”, n, m);
    return 0;
}
```

Read/write data from the standard input/output (cond.)

- Common parameters for the **format.**

```c
#include <stdio.h>
int main() {
    // Read/write a character
    char c;
    scanf("%c", &c);
    printf("%c\n", c);
    // Read a floating number, and write it with 2 decimals
    float f;
    scanf("%f", &f);
    printf("%.2f\n", f);
    // Read/write a string of characters
    char str[10];
    scanf("%s", str);
    printf("%s\n", str);
    return 0;
}
```

Command line arguments

- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program.

```
#include <stdio.h>
int main( int argc, char *argv[] ) {
        if( argc == 2 ) {
                printf("The argument supplied is %s\n", argv[1]);
        }
        else if( argc > 2 ) {
                printf("Too many arguments supplied.\n");
        }
        else {
                printf("One argument expected.\n");
}}
```

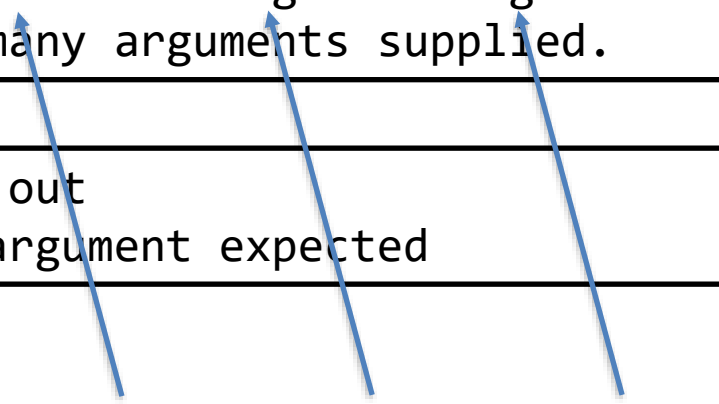
Command line arguments

- It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and *argv[n] is the last argument.

```
$./a.out testing
The argument supplied is testing
```

```
$./a.out testing1 testing2
Too many arguments supplied.
```

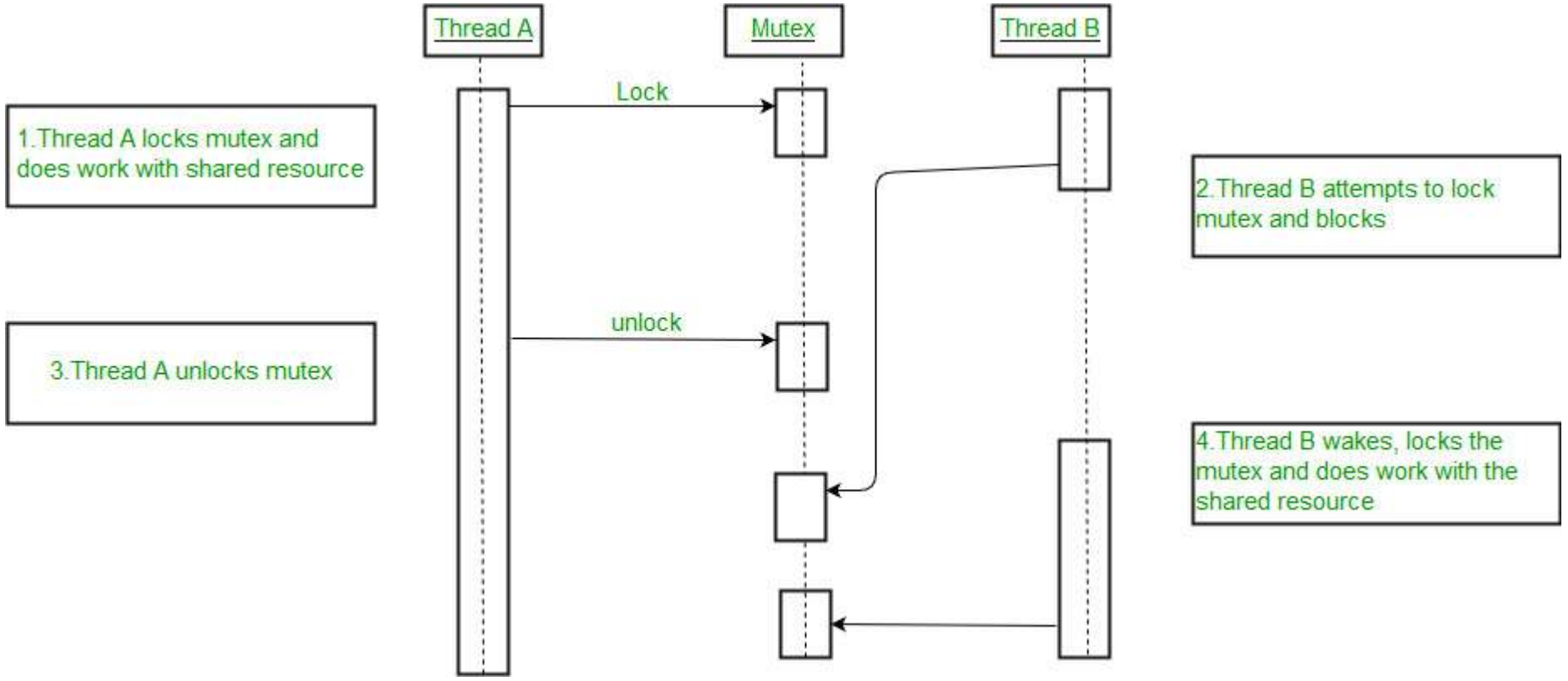
```
$./a.out
One argument expected
```



Pthread library

- Thread creation
- Mutex operation

Use case: using mutex to avoid race condition



Example code: creation and passing arguments into threads

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
// needed header files

pthread_t tid[2];
int counter;
// defines the global variables.

struct thread_args
{
    int thread_id;
    int count_times;
};
// defines the thread arguments
```

```c
void *trythis(void *arg)
{// cast the argument into specified format
    struct thread_args *parsed_args =
           (struct thread_args *)arg;

    printf("\n Job %d has started\n",
           parsed_args->thread_id);

    for (int i = 0; i < parsed_args->count_times; i++)
    {
        counter += 1;
    }
    return NULL;
}
```

Example code: parsing the command line arguements

```
int main(int argc, char *argv[])
{
    int error;
    int thread_id = 0;

    if (argc != 3)
    {
        printf("wrong arg counts");
        return -1;
    }
    int op_count[2] = {atoi(argv[1]), atoi(argv[2])};
    for (int i = 0; i < 2; i++)
    {
        printf("add %d times in thread %d\n", op_count[i], i);
    }
```

Example code, create the threads and wait till it ends.

```
    struct thread_args arg_list[2];

    while (thread_id < 2)
    {
        arg_list[thread_id].thread_id = thread_id;
        arg_list[thread_id].count_times = op_count[thread_id];
        error = pthread_create(&(tid[thread_id]), NULL, &trythis, arg_list[thread_id]);
        if (error != 0)
            printf("\nThread can't be created :[%s]", strerror(error));
        thread_id++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    printf("Global value counter's value is : %d\n", counter);
    return 0;
}
```

Allocate/release memory

- In C programming, the following functions are used to allocate bytes of uninitialized storage.

void* malloc(size_t size);

```c
#include <stdio.h>
int main() {
    // Define a double pointer pointing to the address of a matrix
    int** mat;
    // Create an array of pointers
    mat = (int **)malloc(10 * sizeof(int *));
    // Create an array of integers for each pointer;
    for (int i = 0; i < 10; ++ i) {
        mat[i] = (int *)malloc(10 * sizeof(int));
    }
    // Please note that the value of mat[i][j] is uninitialized
    // Directly accessing mat[i][j] may encounter unexpected errors
    return 0;
}
```

Allocate/release memory

- In C programming, the following functions are used to allocate memory for an array of objects and initialize all bytes in the allocated storage to *zero*.

void* calloc(size_t num, size_t size);

```
#include <stdio.h>
int main() {
    // Define a double pointer pointing to the address of a matrix
    int** mat;
    // Create an array of pointers
    mat = (int **)calloc(10, sizeof(int *));
    // Create an array of integers for each pointer;
    for (int i = 0; i < 10; ++ i) {
        mat[i] = (int *)calloc(10, sizeof(int));
    }
    // Please note that now all mat[i][j] is zero
    return 0;
}
```

Allocate/release memory

- In C programming, the following functions are used to deallocate the space previously allocated by **malloc(), calloc()**.

void* free(void* ptr);

```
#include <stdio.h>
int main() {
    // Define a double pointer pointing to the address of a matrix
    int** mat;
    // Allocate memory using malloc() or calloc()
    ...
    // First release memory allocated for each pointer
    for (int i = 0; i < 10; ++ i) {
        free(mat[i]);
    }
    // Then release memory allocated for the pointer array
    free(mat);
    return 0;
}
```

Environment & Evaluation

- IDE
- Compile the C program
- Evaluate by the Linux pipe

IDE & Terminal

- Install Visual Studio Code
- Build your remote server

Install Visual Studio Code

- Visual Studio Code is an open-source IDE that runs on all platforms, for classmates which are not that familiar with the vim, we recommend you VSC as the IDE
- You can download the VSC in the following link

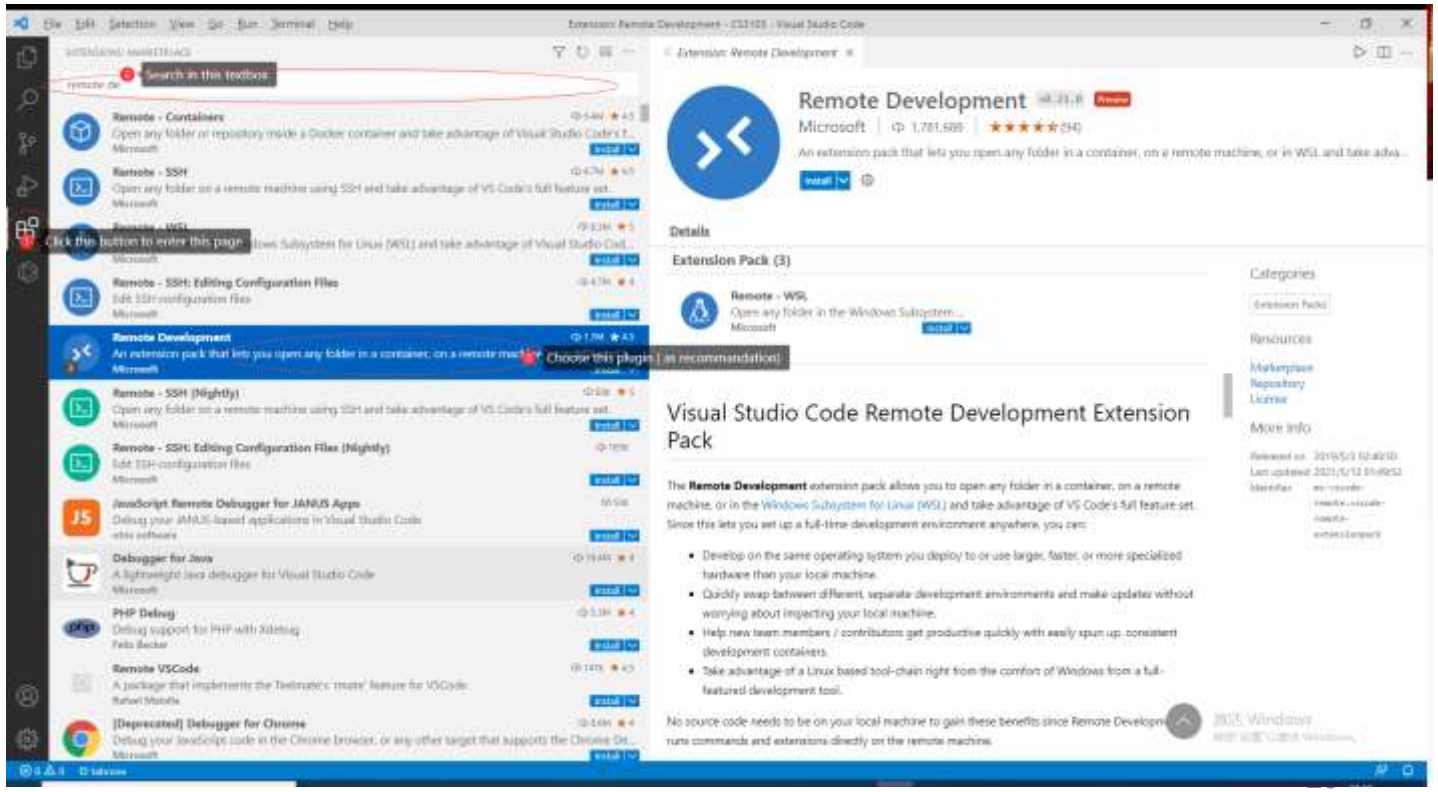
  https://code.visualstudio.com/download
- As recommendation, please install on your own computer instead of the CSLAB computers.

Build your remote environment: install plugins

- Please read the following document if you have any problem.

  https://code.visualstudio.com/docs/remote/remote-overview

- We can use VSC to build the system in remote
- Install the extension in VSC
  - We use the official remote plugin



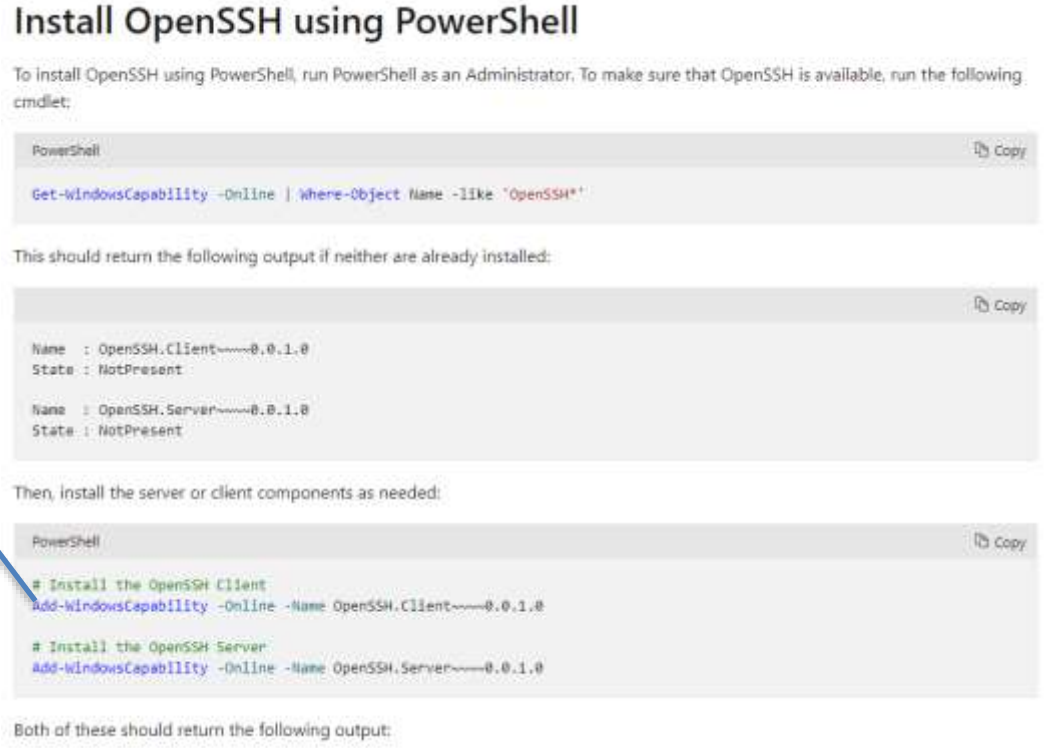
Install Open SSH Client

- For those people who have his/her first trying on Windows, you may need to install the SSH, find the help in the following link

  https://code.visualstudio.com/docs/remote/ssh#_getting-started

- You can use the PowerShell to install OpenSSH tool suits
  - By running the PowerShell in admin model
  - And type:

    Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.
  - And reboot VSC



Install OpenSSH using PowerShell

To install OpenSSH using PowerShell, run PowerShell as an Administrator. To make sure that OpenSSH is available, run the following cmdlet:

PowerShell Copy

```
Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH*'
```

This should return the following output if neither are already installed:

Copy

```
Name  : OpenSSH.Client~~~~0.0.1.0
State : NotPresent

Name  : OpenSSH.Server~~~~0.0.1.0
State : NotPresent
```

Then, install the server or client components as needed:

PowerShell Copy

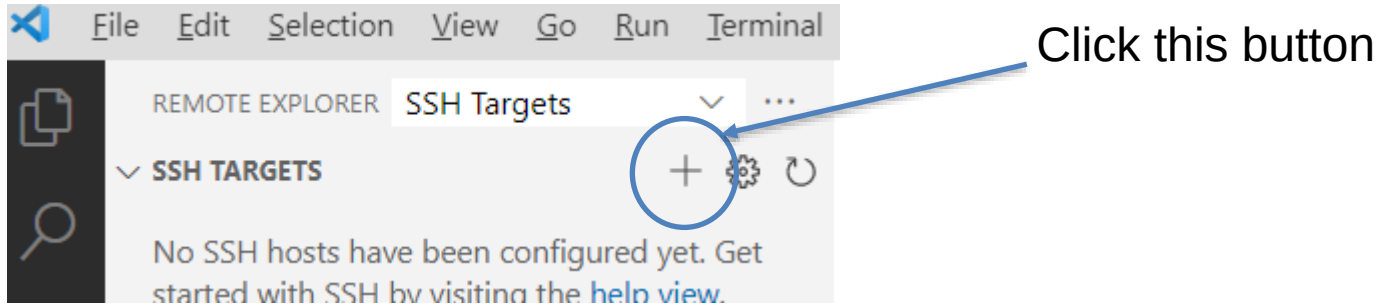
```
# Install the OpenSSH Client
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

# Install the OpenSSH Server
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Both of these should return the following output:

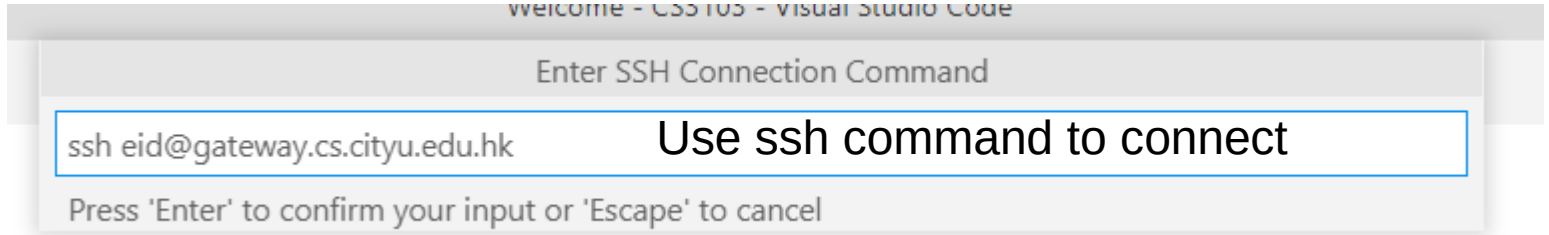
Add the target server

- If you have successfully installed the OpenSSH suit and reboot the VSC, you can add the remote server now.

1



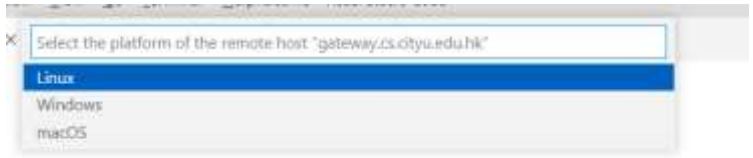
2



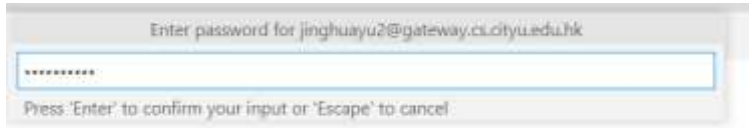
Connect to the remote server

- Then, you can use the gateway server in remote now

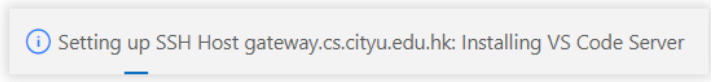
Click the button to create directory in remote server



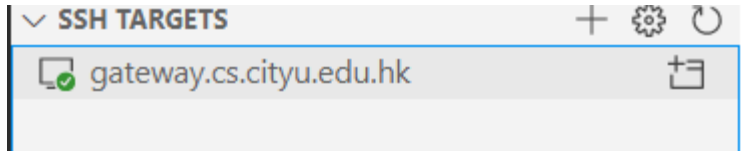
Choose Linux



Type in your password



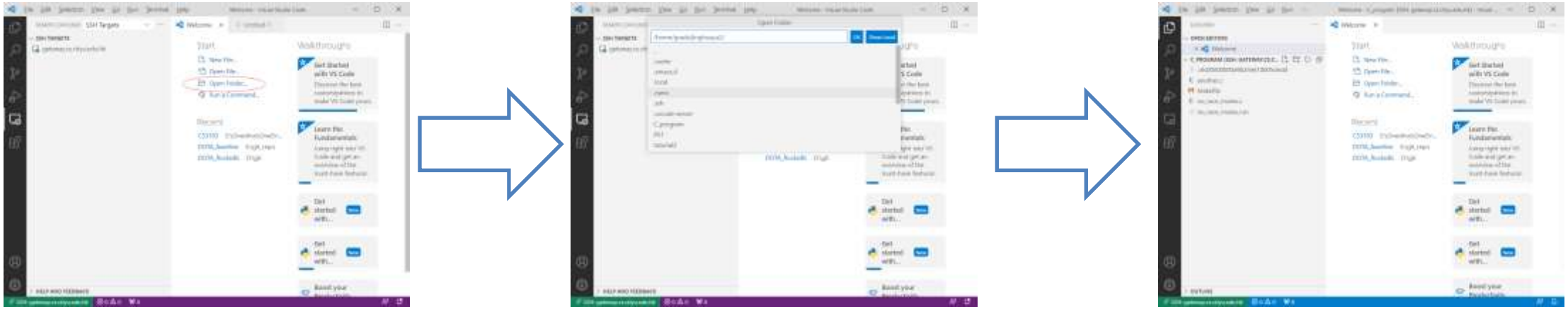
Wait for a second



And you logged in

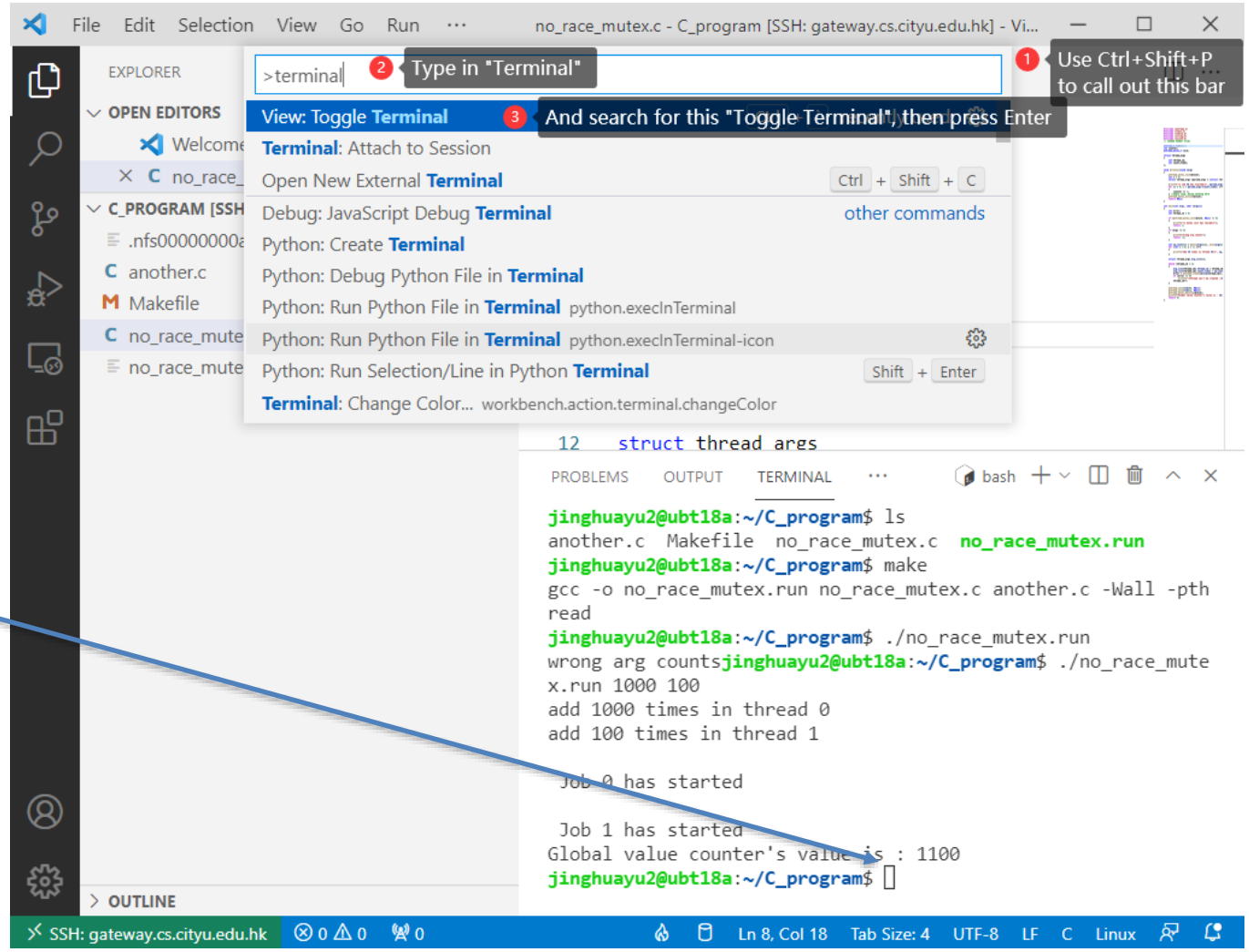
What's Next?

- After you logged in, you can just use VSC as a normal IDE, just like the Clion, Eclipse, CFree or any other IDE you used before.
- First step, use open folder button to enter the work directory.
- Next step, log in again if needed, and then enjoy coding



Install all tools you need

- When you open a .c file at the first time, VSC will ask you to install the extension tool, just click okay and choose the tool you like
  - Highlight, auto completion, syntax check
  - Trigger a remote terminal to get rid of Xterm, you can run command here to compile and run.
- Enjoy your coding, with the help of IDE, coding in C won't that hard to get started.



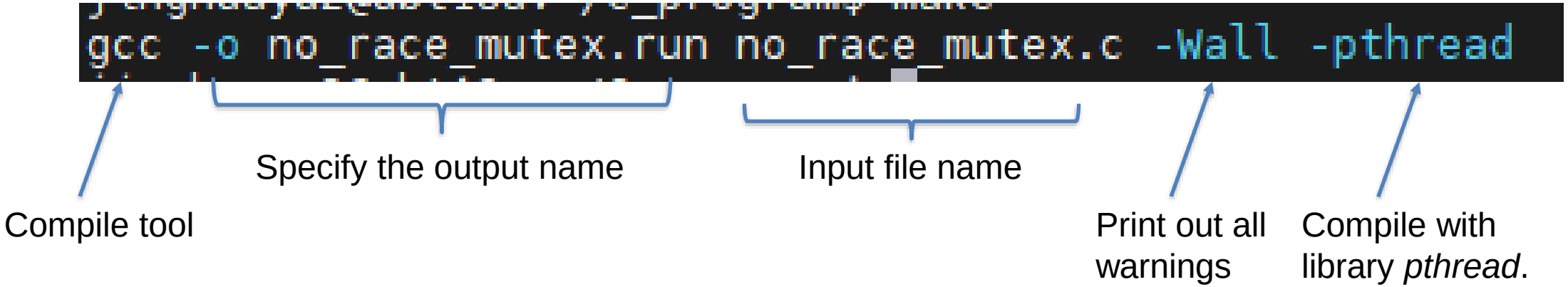
Compile the C program

- Use gcc for a simple project
- makefile

If your file contains only one file, we recommend using the gcc

- For example, you have the file, named *no_race_mutex.c,* and you want to compile it as a runnable file named no_race_mutex.run, you can use the following command.

gcc -o no_race_mutex.run no_race_mutex.c -Wall -pthread



Add if you're build a multi files project, we recommend you make

- In our project, we will provide a basic make file for you, if you don't adding any file into the project, all you need to do is use the *make* command.

```
jinghuayu2@ubt18a:~/C_program$ make
gcc -o no_race_mutex.run no_race_mutex.c -Wall -pthread
jinghuayu2@ubt18a:~/C_program$
```

The make file will generate the compile commands for you.

- Also, you can separate your project into modules, and specify the module you want to build.

```
jinghuayu2@ubt18a:~/C_program$ make no_race_mutex
gcc -o no_race_mutex.run no_race_mutex.c -Wall -pthread
jinghuayu2@ubt18a:~/C_program$
```

Makefile structure

use *make clean* to clean up all files

Shared arguments for GCC

default target of make command

```
FLAGS = -Wall -pthread

all: no_race_mutex

clean:
	rm -f *.run
no_race_mutex: no_race_mutex.c
	gcc -o no_race_mutex.run no_race_mutex.c $(FLAGS)

~
```

Your module name, and the compile command for it.

Evaluate by the Linux Redirection

- Use the Linux redirection to read/write data

Use the Linux redirection to read/write data

- Input/Output (I/O) redirection in Linux refers to the ability of the Linux operating system that allows us to change the standard input (stdin) and standard output (stdout) when executing a command on the terminal.
  - Overwrite the standard input using the '<' symbol.
  - Overwrite the standard output using the '>' symbol.