# Project 1: Realizing Concurrency using Linux Processes and Threads

Qinya Li
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Spring 2023

# Objective

- Design and develop systems programs using C/C++
- Effectively use Linux system calls for process control and management, especially, *fork, exec, wait, pipe and kill system call API*.
- Concurrent execution of processes.
- Use Posix Pthread library for concurrency.
- Use graphic tools or program to analyze data.

# Project 1

- Copy
- Shell
- Matrix multiplication

# Project 1：Copy

- Copy file from one to another

# Project 1：Copy

- Copy file from one to another
- Use two processes
- Processes communicate using pipe system call

# Project 1：Copy

- Copy file from one to another
- Use two processes
- Processes communicate using pipe system call
- Use buffer in order to copy faster

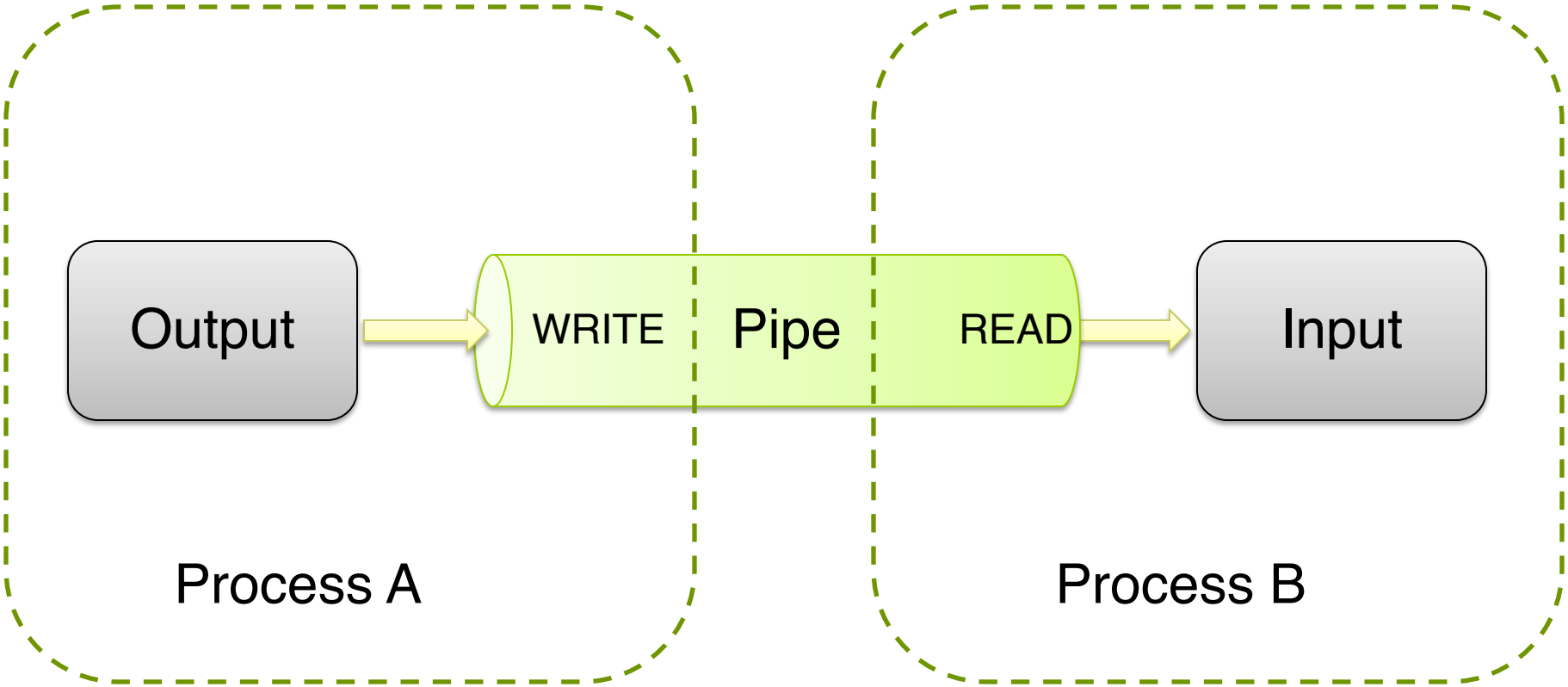
# Project 1：Copy

- Copy file from one to another
- Use two processes
- Processes communicate using pipe system call
- Use buffer in order to copy faster
- Use system calls to get executing time

# Project 1：Copy

- Copy file from one to another
- Use two processes
- Processes communicate using pipe system call
- Use buffer in order to copy faster
- Use system calls to get executing time
- Use graphic tools or programs to analyze the result

# Ordinary Pipe



# Project 1: Copy Tips

- Open a file for reading and check for errors:

```
FILE *src;

// Open source file
src = fopen (argv[1],"r");

// Check for file error
if (src == NULL) {
    printf ("Error: Could not open file '%s'.\n", argv[1]);
    exit(-1);
}
```

  - "r": Opens a source file for reading

# Project 1 Write Tips

- Open a file for writing and check for errors:

```
FILE *dest;

// Open destination file
dest = fopen(argv[2], "w+");

// Check for file error
if (dest == NULL) {
    printf("Error: Could not open file '%s'.\n", argv[2]);
    fclose(src);
    exit(-1);
}
```

  - "w+": Opens an empty file for both reading and writing. If the given file exists, its contents are destroyed.

# Project 1 Tips

- Read and write to a file
  - Read/Write a block of chars:

    **size_t fread( void ***_buffer_**, size_t** _size_**, size_t** _count_**, FILE ***_stream_ **);**

    - **fread** returns the number of full items actually read

    **size_t fwrite( const void ***_buffer_**, size_t** _size_**, size_t** _count_**, FILE ***_stream_ **);**

    - **fwrite** returns the number of full items actually written

  - Read an individual char:

    **int fgetc( FILE ***_stream_ **);**

    **int fputc( int** _c_**, FILE ***_stream_ **);**

# Project 1 Tips

- Don't forget to check for EOF (with fgetc) or size of buffer written (with fread/fwrite)
- Don't forget to close your file with fclose

# Project 1 Problem 1 Tips

- Timing I

```
clock_t start, end;
double elapsed;
start = clock();
……
end = clock();
elapsed = ((double) (end - start)) /
                CLOCKS_PER_SEC * 1000;
printf("Time used: %f millisecond\n", elapsed);
```

# Project 1 Problem 1 Tips

- Timing II

```
struct timeval startTime, endTime;
struct timezone tz;
struct tm *tm;
gettimeofday(&startTime, &tz);
……
gettimeofday(&endTime, &tz);
long run_time_in_microseconds;
run_time_in_microseconds = endTime.tv_usec - startTime.tv_usec;
printf("Executable time for quick sort single threaded is %d
microseconds.\n",run_time_in_microseconds);
```

```
struct timeval{
long int tv_sec; // seconds
long int tv_usec; // microseconds
}
```

# Using fork()

- Fork the child process:

```
pid_t ForkPID;
ForkPID = fork();
```

- Write a quick switch statement:

```
switch (ForkPID) {
    // -1, fork failure
    case -1:
            printf("Error: Failed to fork.\n"); break;
    // 0, this is the child process
    case 0:
            ……
            break;
    // > 0, parent process and the PID is the child's PID
    default:
            ……
}
```

# Using Pipe – Part 1

- First, create a pipe and check for errors

```
int mypipe[2];
if (pipe(mypipe)) {
    fprintf (stderr, "Pipe failed.\n");
    return -1;
}
```

- Second, fork your processes
- Third, close the pipes you don't need in that process
  - reader should close(mypipe[1]);
  - writer should close(mypipe[0]);

# Using Pipe – Part 2

- Fourth, the writer should write the data to the pipe
  - ssize_t write(int pipe_id, const void *buf, size_t count)
  - write(mypipe[1],&c,1);
- Fifth, the reader reads from the data from the pipe:
  - ssize_t read(int pipe_id, const void *buf, size_t count)
  - while (read(mypipe[0],&c,1)>0) {

  //do something, loop will exit when WRITER closes pipe }
- Sixth, when writer is done with the pipe, close it
  - close(mypipe[1]); //EOF is sent to reader
- Fifth, when reader receives EOF from closed pipe, close the pipe and exit your polling loop
  - close(mypipe[0]); //all pipes should be closed now

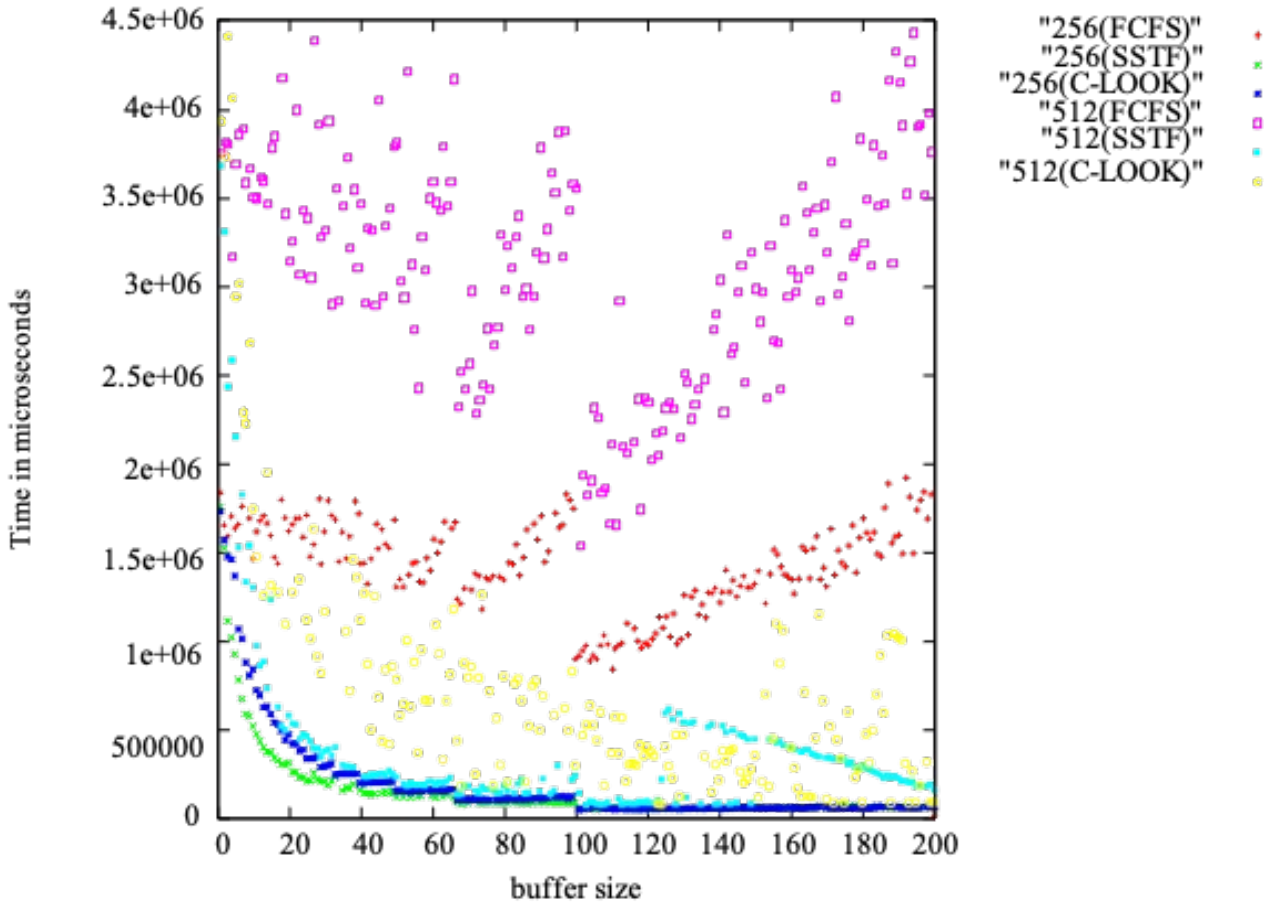
# Example

```
lqy@ubuntu: ~/OSProject
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~/OSProject$ gcc copy.c -o copy
lqy@ubuntu:~/OSProject$ ls
copy  copy.c  copy.out  dest.txt  sample_data.in  src.txt
lqy@ubuntu:~/OSProject$
```

```
lqy@ubuntu: ~/OSProject
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~/OSProject$ ./copy src.txt dest.txt 100
Read file end.
Write file end.
Time used (PipeCopy): 0.384000 millisecond.
lqy@ubuntu:~/OSProject$ ./copy src.txt dest.txt 1000
Read file end.
Write file end.
Time used (PipeCopy): 0.248000 millisecond.
lqy@ubuntu:~/OSProject$
```

# Gnuplot

- Use graphic tools or program to analyze data
- http://www.gnuplot.info/



# Project 1: Shell

- Write a shell-like program as a server

# Project 1: Shell

- Write a shell-like program as a server
- Handle the commands with arguments and the commands connected by pipes

# Project 1: Shell

- Write a shell-like program as a server
- Handle the commands with arguments and the commands connected by pipes
- Try to support more than one client

# Project 1: Shell

- Write a shell-like program as a server
- Handle the commands with arguments and the commands connected by pipes
- Try to support more than one client
- Do NOT use the linux shell command line processor to run your command line, you should handle the arguments by yourself

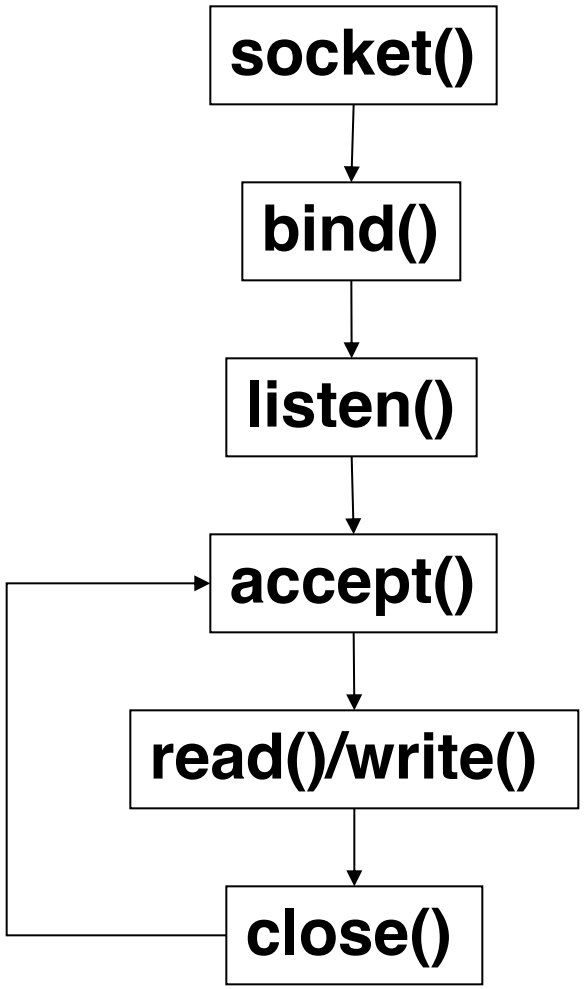
# Resources

- Here are a few good pages that can help:
  - http://www.linuxhowtos.org/C_C++/socket.htm
  - http://en.wikipedia.org/wiki/Internet_socket
  - http://en.wikipedia.org/wiki/Berkeley_sockets
- Sample client/server codes available on course webpage
- You can use this code on these pages, as you will need to change them to suit your own purposes

# Socket Programming

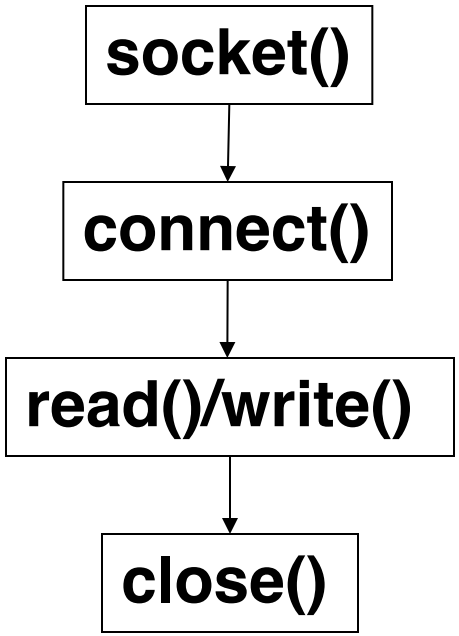
# Steps to Create Server Side

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call.
3. Listen for connections with the listen() system call
4. Accept a connection with the accept() system call (This call typically blocks until a client connects with the server)
5. Send and receive data with read() and write() system calls
6. Close connection with close() system call

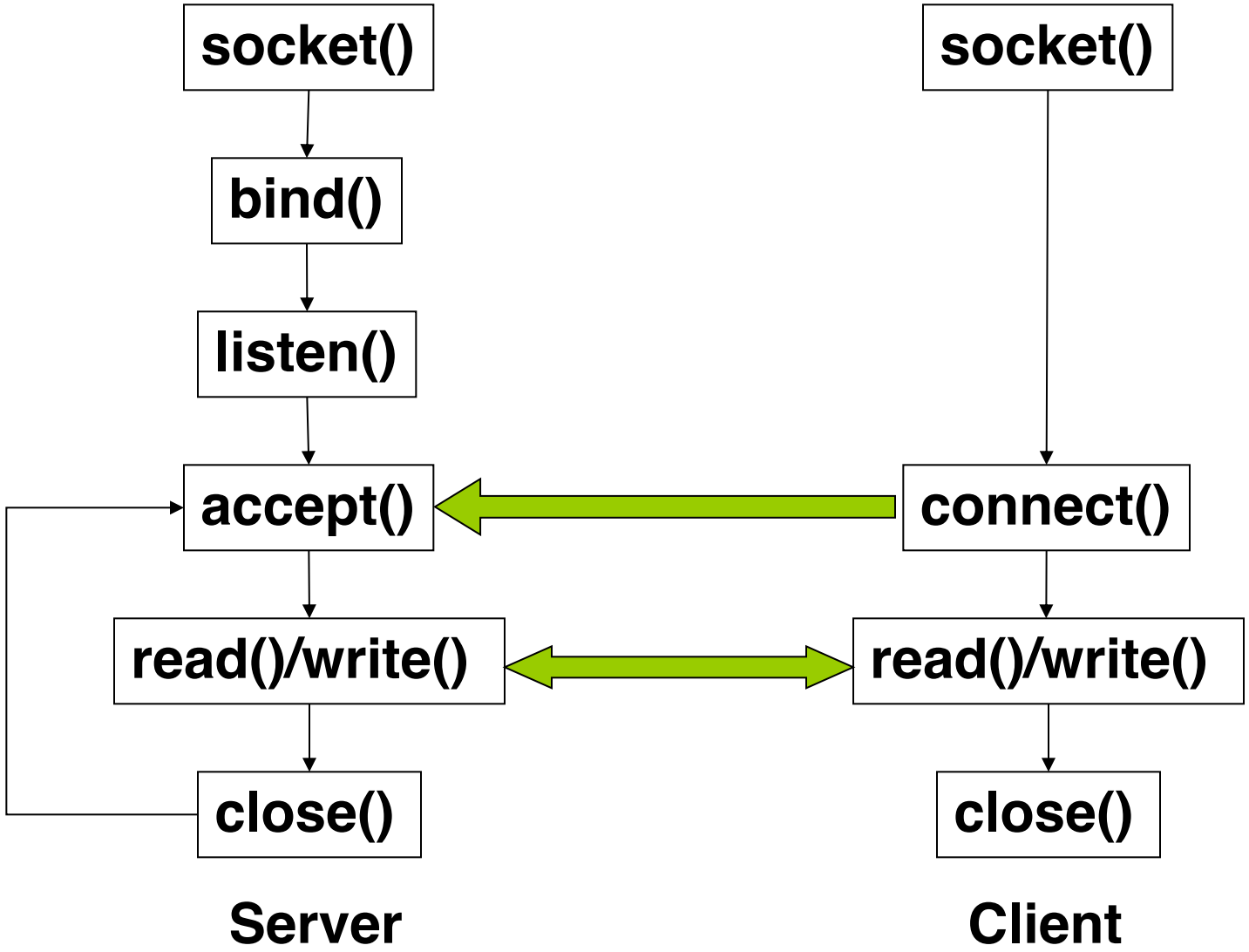


# Steps to Create Client Side

1. Create a socket with the socket() system call
2. Connect the socket to the address of the server using the connect() system call
3. Send and receive data with read() and write() system calls.
4. Close the socket with close() system call



# Interaction Between Client and Server



# Internet Domain Socket

- IP address:
  - 32 bits (IPv4) or 128 bits (IPv6)
  - C/S work on same host: just use localhost
- Port
  - 16 bit unsigned integer
  - Lower numbers are reserved for standard services
- Transport layer protocol: TCP / UDP

# Headers

- #include <stdio.h>
- #include <stdlib.h>
- #include <string.h>
- #include <sstream>
- #include <unistd.h>
- #include <sys/types.h>
  - Definitions of a number of data types used in system calls
- #include <sys/socket.h>
  - Definitions of structures needed for sockets
- #include <netinet/in.h>
  - Constants and structures needed for internet domain addresses

# Creating Socket

```
int sockfd
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
    perror("ERROR opening socket");
    exit(2);
}
```

- AF_INET: address domain
- SOCK_STREAM: stream socket, characters are read in a continuous stream as if from a file or pipe
- 0: protocol. The operating system chooses the most appropriate protocol. It will choose TCP for stream sockets.

# Binding Socket

```
struct sockaddr_in serv_addr;
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(BASIC_SERVER_PORT);
bind(sockfd, (sockaddr*) &serv_addr, sizeof(serv_addr));
    //error check
```

- INADDR_ANY: get IP address of the host automatically
- htonl, htons: data format conversion
- bind(): binds a socket to an address

# Listening and Accepting Connection

```
listen(sockfd, 5);
```

- listen(): allows the server to listen on the socket for connections, with a backlog queue of size 5.

```
int client_sockfd;
struct sockaddr_in client_addr;
int len = sizeof(client_addr);
client_sockfd = accept(sockfd, (sockaddr *) &client_addr,
&len);
    //error check
```

- accept(): block process until a client connects to the server. It returns a new socket file descriptor, if the connection is created.

# Reading and Writing

```
char buf[1024];
int nread = read(client_sockfd, buf, 1024);
```

read(): reads from the socket

```
write(client_sockfd, buf, len);
```

write(): writes to the socket

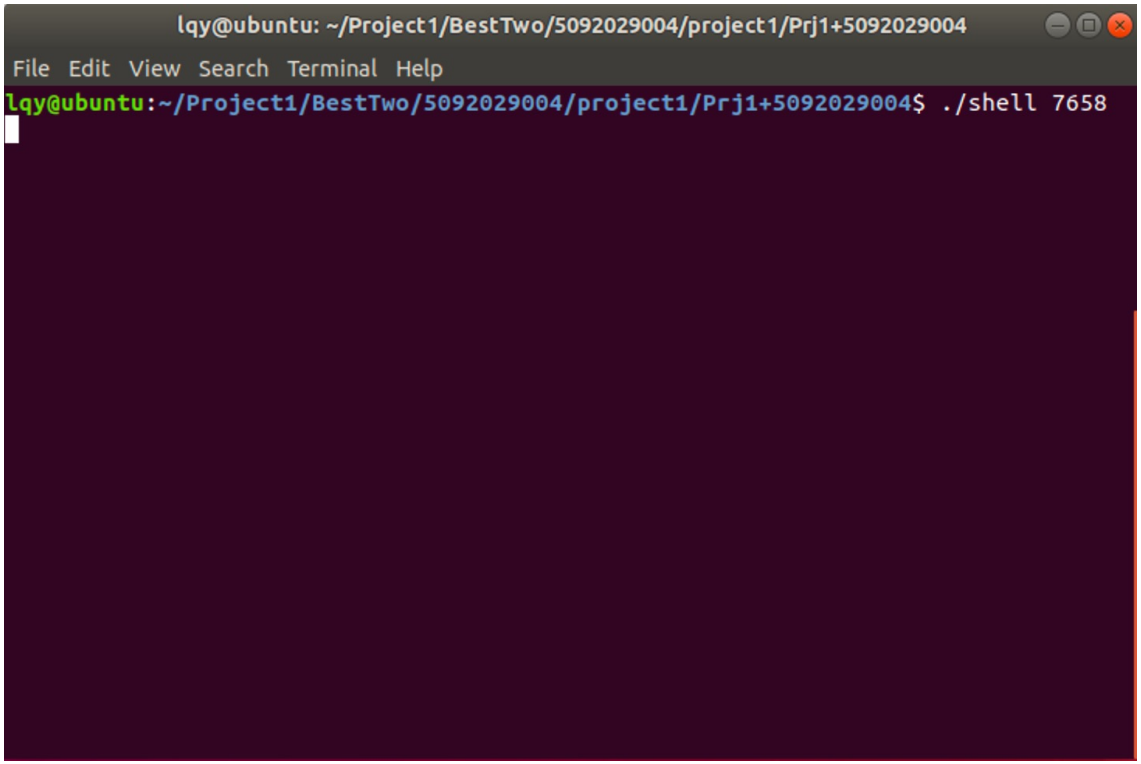
```
close(client_sockfd);
```

close(): closes the socket

# Connecting A Client to A Server

```
int sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // error check
struct sockaddr_in serv_addr;
struct hostent *host;
serv_addr.sin_family = AF_INET;
host = gethostbyname(argv[1]);
    // error check
memcpy(&serv_addr.sin_addr.s_addr, host->h_addr,
    host->h_length);
serv_addr.sin_port = htons(BASIC_SERVER_PORT);
connect(sockfd, (sockaddr *) &serv_addr, sizeof(serv_addr))
    // error check
```

# Example


lqy@ubuntu: ~/Project1/BestTwo/5092029004/project1/Prj1+5092029004
File Edit View Search Terminal Help
lqy@ubuntu:~/Project1/BestTwo/5092029004/project1/Prj1+5092029004$ ./shell 7658

# Example

```
lqy@ubuntu: ~/Project1/BestTwo/5092029004/project1/Prj1+5092029004
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~/Project1/BestTwo/5092029004/project1/Prj1+5092029004$
lqy@ubuntu:~/Project1/BestTwo/5092029004/project1/Prj1+5092029004$ ./shell 7658
New child process (seq=1)
```

```
lqy@ubuntu: ~
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~$ telnet localhost 7658
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome to my shell!
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004$
```

# Example

```
lqy@ubuntu: ~/Project1/BestTwo/5092029004/project1/Prj1+5092029004
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~/Project1/BestTwo/5092029004/project1/Prj1+5092029004$ ./shell 7658
New child process (seq=1)
New child process (seq=2)
```

```
lqy@ubuntu: ~
File  Edit  View  Search  Terminal  Help
lqy@ubuntu:~$ telnet localhost 7658
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome to my shell!
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004$
```

# Project 1 Shell Tips – Part 1

- Shell must specify an input loop as described, so you should implement a prompt-input-execute loop

- Sample test run:

```
lqy@ubuntu: ~
File  Edit  View  Search  Terminal  Help
src.txt
test
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004$ pwd
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004$ ls -l
total 252
-rw-rw-r-- 1 lqy lqy  2152 Jun 19  2012 copy.c
-rw-rw-r-- 1 lqy lqy 15770 Jun 19  2012 copy.ods
-rwxrwxr-x 1 lqy lqy 13008 Feb 16 22:42 copy.out
-rw-rw-r-- 1 lqy lqy    86 Feb 16 22:44 des.txt
-rw-rw-r-- 1 lqy lqy   460 Jun 19  2012 Makefile
-rwxrwxr-x 1 lqy lqy 12313 Jun 19  2012 multi
-rwxrwxr-x 1 lqy lqy   314 Jun 19  2012 multi-benchmark.sh
-rw-rw-r-- 1 lqy lqy  5988 Jun 19  2012 multi.c
-rw-rw-r-- 1 lqy lqy 38612 Jun 19  2012 multi.ods
-rw-rw-r-- 1 lqy lqy 83673 Jun 19  2012 Performance Analysis.pdf
-rw-rw-r-- 1 lqy lqy   476 Jun 19  2012 Prj1README
-rwxrwxr-x 1 lqy lqy 13712 Feb 19 19:31 shell
-rw-rw-r-- 1 lqy lqy  6236 Jun 19  2012 shell.c
-rwxrwxr-x 1 lqy lqy 13712 Feb 17 02:18 shell.out
-rw-rw-r-- 1 lqy lqy  3828 Jun 19  2012 single.c
-rw-rw-r-- 1 lqy lqy    86 Feb 16 22:43 src.txt
-rw-rw-r-- 1 lqy lqy  1772 Jun 19  2012 test
/home/lqy/Project1/BestTwo/5092029004/project1/Prj1+5092029004$
```

# Project 1 Shell Tips – Part 2

- Fork a child process for executing each command (the exec system call will overlay the child code and use the child process space)
- Sample code run in the child process:

```
if (execvp(command_array[0],command_array) == -1) {
    printf("Error: running command: '%s'\n", line);
    exit(0);}
```

For example:

line = “ls –l”

command_array[0] = “ls”

command_array[1] = “-l”

command_array[2] = NULL

# Project 1 Shell Tips – Part 3

- Parsing command

```
int parseLine(char *line, char *command_array[]) {
    char *p;
    int count=0;
    p = strtok(line, " ");
    while (p && strcmp(p,"|")!=0) {
      command_array[count] = p;
      count++;
      p = strtok(NULL," ");
    }
    return count;
}
```

# Project 1 Shell Tips – Part 4

- Very IMPORTANT: you do not need to handle “*” or “?”.
- Handling “exit” command!

# Project 1 Shell Tips – Part 5

- In the parent process, wait for a child process to complete (exit) before prompting again:

```
// wait (or block current process) until a forked child finished
int r;
wait(&r);
```

An alternative way (or see **waitpid() call):**

```
// do not exit until ALL child processes are finished
// be aware, this may behave differently depending on the
// machine you use or how fast things are
while (wait(&status) != 0) {}
```

# Project 1 Shell Tips – Part 6

- Needs to add:
  - Processing the pipe “l” symbol in the command line
  - Piping/redirecting data from one process to another
- If pipe without command after it?
  - throw an error message, continue loop
- If pipe at front of command line?
  - throw an error message, continue loop

# Project 1 Shell Tips – Part 7

- To “redirect” input/output, use dup command
  - We prefer to use **dup2() command**
  - dup2(int oldfd, int newfd);
- Steps for programming:
  - First, have main program process both sides of the pipe into a command array
    - create a pipe (just like in problem 1)
    - fork a child process A
    - fork a child process B
    - close both ends of the pipe in the parent

# Project 1 Shell Tips – Part 8

- Steps for programming, cont.:
  - Second, in child process A
    - close the READ pipe, and standard output
      - **close (mypipe[0]);**
      - **close(1);**
    - dup2 the WRITE pipe onto standard output AND THEN close the WRITE pipe
      - **dup2(mypipe[1],STDOUT_FILENO);**
      - **close (mypipe[1];)**
    - exec the command after pipe(with arguments)

```
if (execvp(command_arrayB[0],command_arrayB) == -1)
    { printf("Error: running command: '%s'\n", line); }
```

# Project 1 Shell Tips – Part 9

- Steps for programming, cont.:
  - Third, in child process B
    - close the WRITE pipe, and standard input
      - **close (mypipe[1]);**
      - **close(0);**
    - dup2 the READ pipe onto standard input AND THEN close the READ pipe
      - **dup2(mypipe[0],STDIN_FILENO);**
      - **close (mypipe[0];)**
    - exec the command  before pipe (with arguments)

```
if (execvp(command_arrayA[0],command_arrayA) == -1)
   { printf("Error: running command: '%s'\n",line); }
```

# Project 1 Shell Tips – Part 10

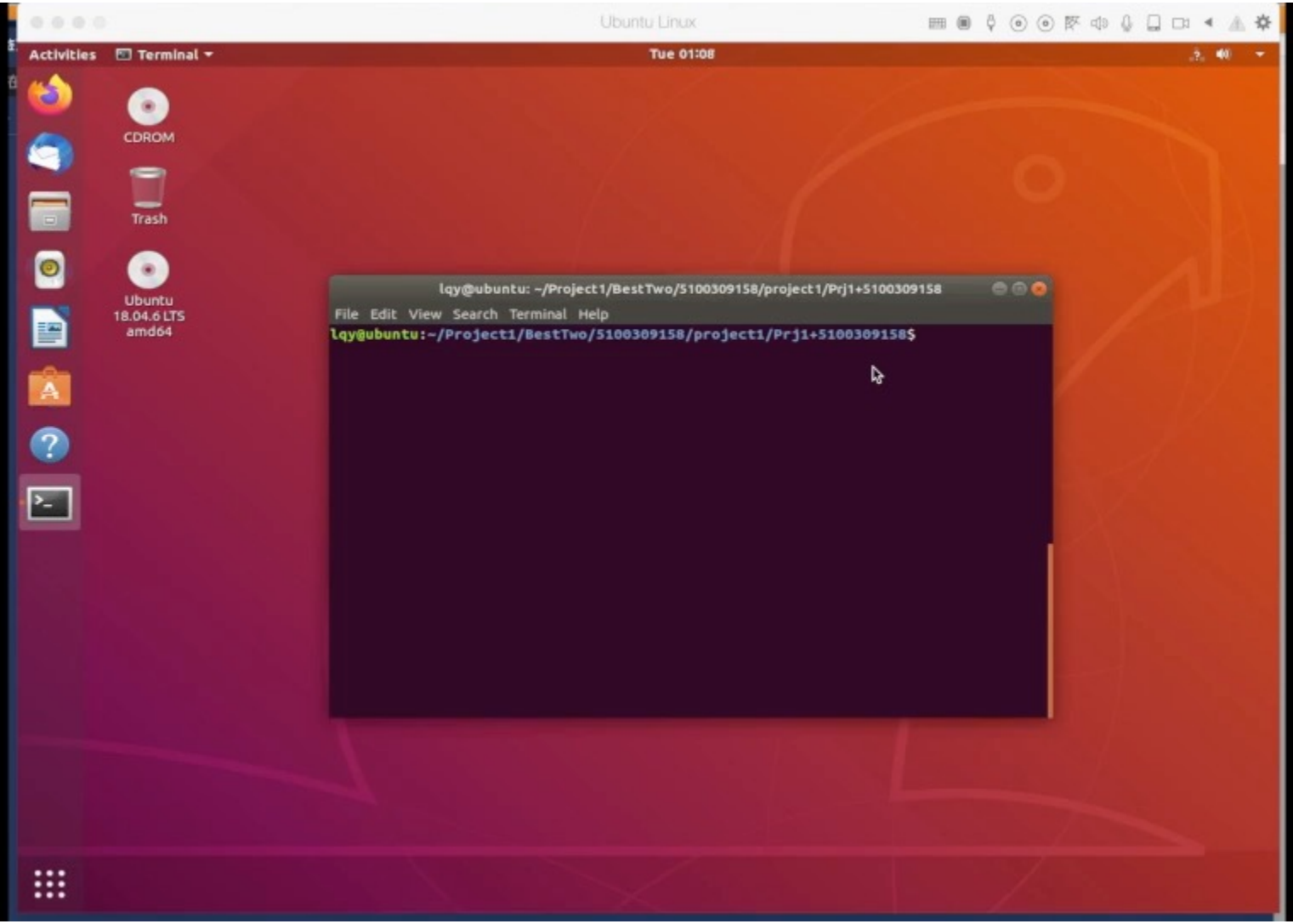
- Steps for programming, cont.:
  - Finally, have the parent wait until child processes are all done.
    - **wait(&r);** // hold back new prompt until old command finishes

# Project 1 Shell NoNo's

- Do NOT use the linux shell command line processor to run your command line. You will get zero points for that solution.

- For example, do not do the following to run a "ls -l *.c" command:
  - **execl("/usr/bin/sh", "sh", "-c", "ls -l *.c", NULL);**
  - This is using the shell "sh" to do all the heavy work and thus is not acceptable

# Sample Shell



# Project 1: Matrix multiplication

- Find a algorithm of matrix multiplication

# Project 1: Matrix multiplication

- Find a algorithm of matrix multiplication
- Implement a single-thread program and a multi-thread program

# Project 1: Matrix multiplication

- Find a algorithm of matrix multiplication
- Implement a single-thread program and a multi-thread program
- Compare the performance of different implementations

# Project 1 Pthread 1

- Header needed:

```
#include <pthread.h>
```

- Preparation:

```
// create a thread 't1'
pthread_t  t1;
pthread_attr_t  attr1;
pthread_attr_init(&attr1);
```

# Project 1 Pthread 1

- Header needed:

```
#include <pthread.h>
```

- Preparation:

```
// create a thread 't1'
pthread_t  t1;
pthread_attr_t  attr1;
pthread_attr_init(&attr1);
pthread_attr_setdetachstate(&attr1,
   THREAD_CREATE_JOINABLE);
```

THREAD_CREATE_JOINABLE
THREAD_CREATE_DETACHED

```
typedef struct
{
        int __detachstate;
        int __schedpolicy;
        struct sched_param __schedparam;
        int __inheritsched;
        int __scope;
        size_t __guardsize;
        int __stackaddr_set;
        void* __stackaddr;
        size_t __stacksize;
} pthread_attr_t;
```

# Project 1 Pthread 1

- Header needed:

```
#include <pthread.h>
```

- Preparation:

```
// create a thread 't1'
pthread_t  t1;
pthread_attr_t  attr1;
pthread_attr_init(&attr1);
pthread_attr_setdetachstate(&attr1,
   THREAD_CREATE_JOINABLE);
```

THREAD_CREATE_JOINABLE
THREAD_CREATE_DETACHED

```
typedef struct
{
        int __detachstate;
        int __schedpolicy;
        struct sched_param __schedparam;
        int __inheritsched;
        int __scope;
        size_t __guardsize;
        int __stackaddr_set;
        void* __stackaddr;
        size_t __stacksize;
} pthread_attr_t;
```

# Project 1 Pthread 1

- Header needed:

```
#include <pthread.h>
```

- Preparation:

```
// create a thread 't1'
pthread_t  t1;
pthread_attr_t  attr1;
pthread_attr_init(&attr1);
pthread_attr_setdetachstate(&attr1,
   THREAD_CREATE_JOINABLE);
pthread_attr_setscope(&attr1, THREAD_SCOPE_SYSTEM);
```

```
typedef struct
{
        int __detachstate;
        int __schedpolicy;
        struct sched_param __schedparam;
        int __inheritsched;
        int __scope;
        size_t __guardsize;
        int __stackaddr_set;
        void* __stackaddr;
        size_t __stacksize;
} pthread_attr_t;
```

# Project 1 Pthread 2

- Prepare thread arguments

```
// create an argument structure (specified before
   this point)
struct  thread_arguments  my_arguments1;
```

- The structure of argument

```
struct  thread_arguments
{
    int first_paramter;
    int second_paramter;
    int return_value;
};
```

# Project 1 Pthread 3

- Launch a thread

```
rc = pthread_create (&t1, &attr1, my_function, &my_arguments1);
if (rc) {
    printf("ERROR; return code from pthread_create(t1) is %d\n", rc);
    exit(-1);}
```

- Sample function for a thread

```
void *my_function (void *args)  {
        thread_arguments *a;                // These two lines are for
        a = (thread_arguments *)args;       // coding convenience
        int a1 = a->first_paramter;
        int b1 = a->second_paramter;
        a->return_value = a1 + b1;
        pthread_exit(NULL);   // or you can return values here, but
                              // you are passing a pointer back
                              // so be careful – i.e. No local variables
}
```

# Project 1 Pthread 4

- Wait on a thread's completion

```
// join thread 1 and wait for completion
void* status1;
rc = pthread_join (t1, &status1);
if (rc)
{
   printf("ERROR; return code from pthread_join(t1)
   is %d\n", rc);
   exit(-1);
}
```

# Project 1 Tips

- Input data file: data.in
- Output file: data.out
- Random: random.out

# Some Tips

- You can use “man” if there is a problem with library functions or system calls

# Some Tips

- You can use “man” if there is a problem with library functions or system calls
- Manuals are you friends. Try to use manuals rather than other books

# Some Tips

- You can use “man” if there is a problem with library functions or system calls
- Manuals are you friends. Try to use manuals rather than other books
- Try to use LATEX to write your report, you will fall in love with it

# Some Tips

- You can use “man” if there is a problem with library functions or system calls
- Manuals are you friends. Try to use manuals rather than other books
- Try to use LATEX to write your report, you will fall in love with it
- Enjoy coding, enjoy project