1. (1)
   Virtualization: The operating system is in charge of the illusion that the system has a very large number of virtual CPUs, thus allowing many programs to seemingly run at once.

   Concurrency: When running multiple threads on one CPU, the operating system divides the CPU running time into various periods and assign the periods to each threads and run each thread by turns.

   Persistency: The ability for the hardware and software to store data persistently to avoid data loss.

   (2)
   Virtualization: Chapter 5, 10, 18
   Concurrency: Chapter 4, 8
   Persistency: Chapter 11, 12, 13, 14, 15

2. **Context switch:** The operating system saves a few register values for the currently-executing process and restore for the soon-to-be-executing process. Therefore, the system resumes execution of another process instead of returning to the process that was running. To save the context, the OS will execute assembly code to save the general purpose registers, PC and the kernel stack pointer, and then restore them for the soon-to-be-executing process. When the OS executes a return-from-trap instruction, the soon-to-be-executing process becomes the currently-running process.

3. (1) When fork() is called, the process is interrupted and user mode switches to kernel mode. At the same time, another PCB is created and a new pid of the process is also created. Finally the newly generated process becomes a sub-process of the original one, and the running time of the child process is set to 0. In the parent process, the return value of fork() is the pid of the child process. In the child process, the return value of fork() is 0. The CPU scheduler decides which process to run next, and if the soon-to-be-executed process is different from the original one, it will trigger a context switch.

   (2) The kernel mode will be switched and release the kernel space memory, then releases the memory in user space. If a process is terminated but its parent process doesn't release the corresponding memory resource, it becomes a zombie process. When the parent process calls wait(), it suspends the current process until a sigchild signal is received or the child process is terminated. If the child process is terminated when calling wait(), it would return the status of the child process immediately. When the parent process receives the sigchild signal from the kernel after calling wait(), it will eliminate the child process completely so that no zombie process will remain.

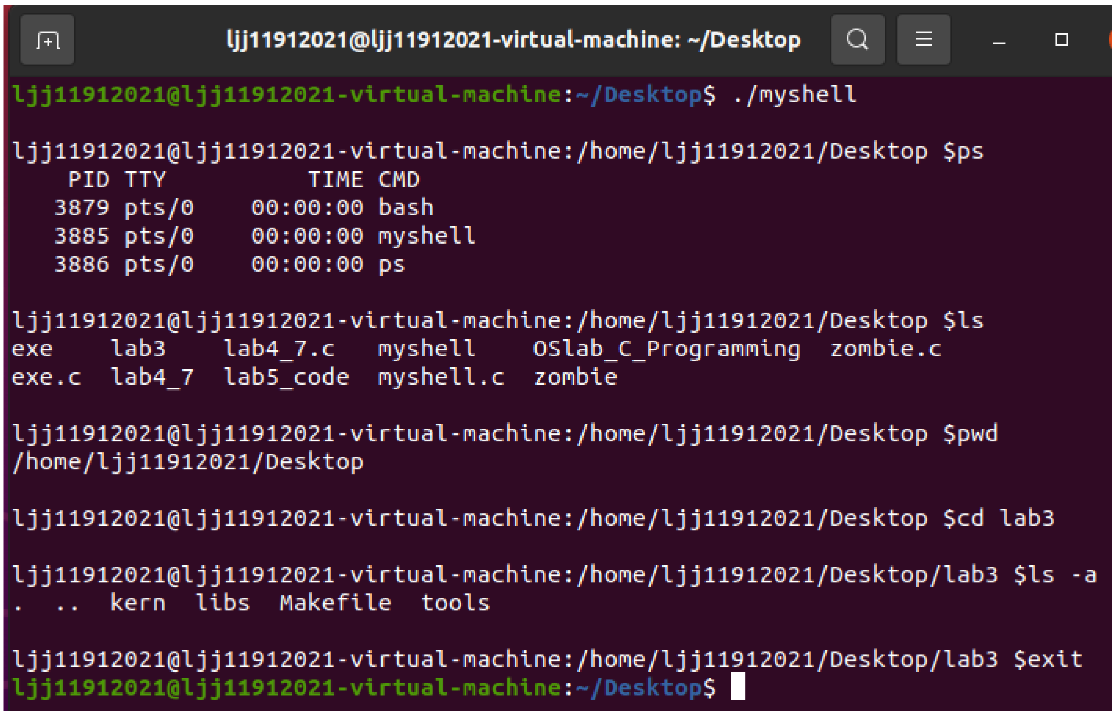
4. System call: Switch user-mode to kernel mode. The system calls are used by user-mode process to operate commands provided by the OS.

Interrupt: As external asynchronous events occur, it sends an interrupt signal to the CPU so that it would pause the current process to switch to another process.

Trap or Exception: Similar to interrupt, but it is the internal synchronous event that triggers the context switch.

5. A process is **new** meaning it is created, and then the process waits for allocating a CPU with **ready** state. It is then **running** as being executed. As a process finishes, it switches to **terminated** state. If a running process is interrupted, it returns to **ready** state. Since one CPU can only run one process at a time, more other processes are at **ready** or **waiting** state. For those at **waiting** state, if a certain condition is fulfilled, it will turn to **ready** state.

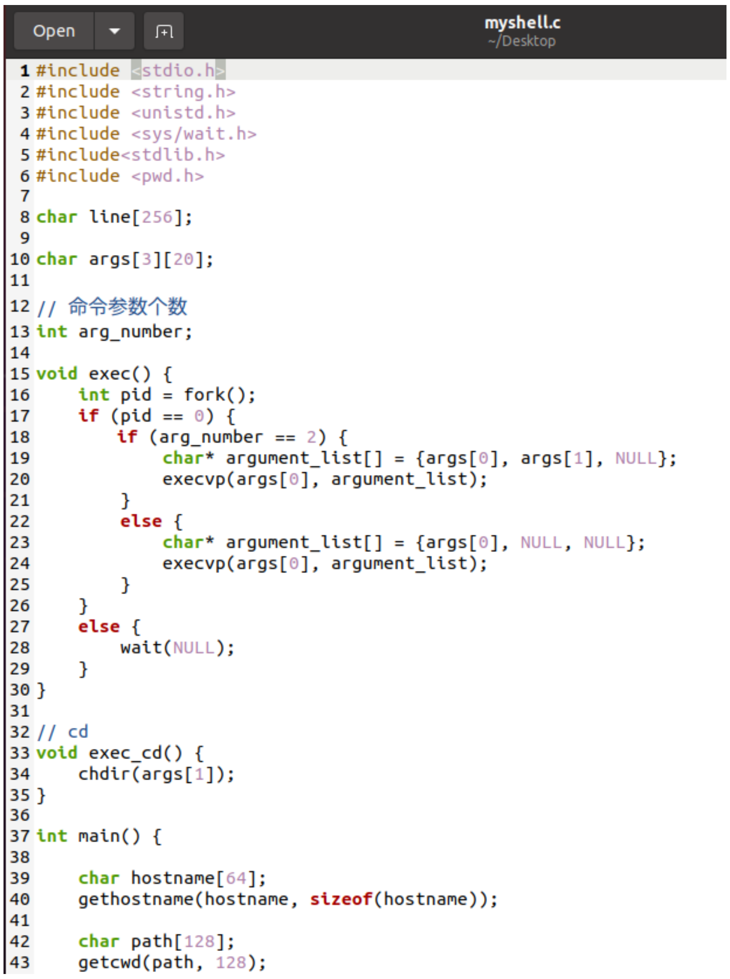
6.



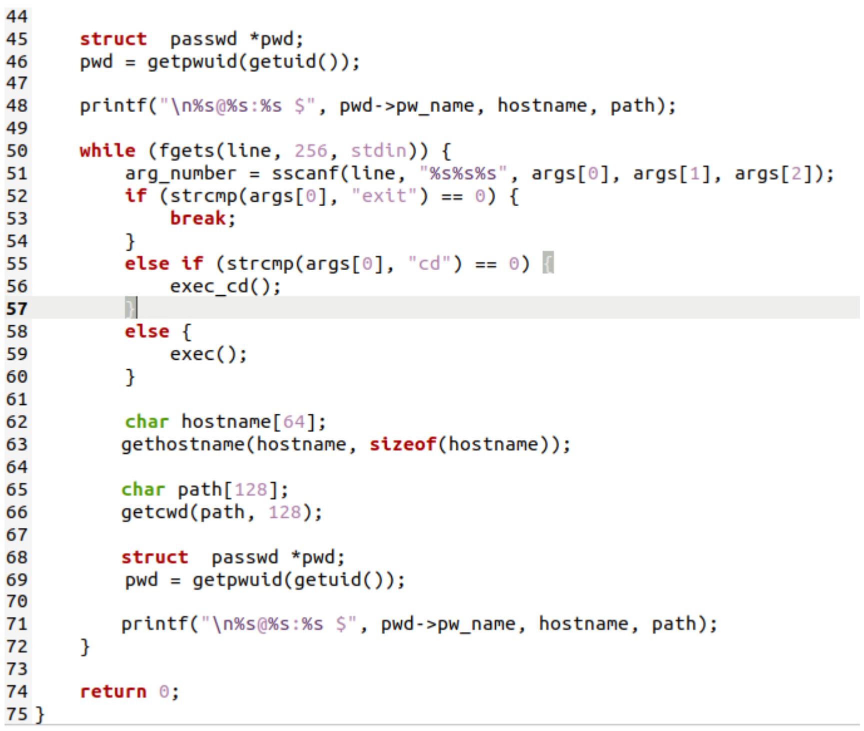
```
ljj11912021@ljj11912021-virtual-machine: ~/Desktop
ljj11912021@ljj11912021-virtual-machine:~/Desktop$ ./myshell

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop $ps
    PID TTY          TIME CMD
   3879 pts/0    00:00:00 bash
   3885 pts/0    00:00:00 myshell
   3886 pts/0    00:00:00 ps

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop $ls
exe    lab3    lab4_7.c   myshell    OSlab_C_Programming  zombie.c
exe.c  lab4_7  lab5_code  myshell.c  zombie

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop $pwd
/home/ljj11912021/Desktop

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop $cd lab3

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop/lab3 $ls -a
.   ..   kern   libs   Makefile   tools

ljj11912021@ljj11912021-virtual-machine:/home/ljj11912021/Desktop/lab3 $exit
ljj11912021@ljj11912021-virtual-machine:~/Desktop$ 
```

Corresponding ouputs of ps, ls, pwd and cd, and ends with "exit"

The program can perform ps, ls, pwd, cd and other basic instructions.



```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include<stdlib.h>
#include <pwd.h>

char line[256];

char args[3][20];

// 命令参数个数
int arg_number;

void exec() {
    int pid = fork();
    if (pid == 0) {
        if (arg_number == 2) {
            char* argument_list[] = {args[0], args[1], NULL};
            execvp(args[0], argument_list);
        }
        else {
            char* argument_list[] = {args[0], NULL, NULL};
            execvp(args[0], argument_list);
        }
    }
    else {
        wait(NULL);
    }
}

// cd
void exec_cd() {
    chdir(args[1]);
}

int main() {

    char hostname[64];
    gethostname(hostname, sizeof(hostname));

    char path[128];
    getcwd(path, 128);
```



```c

    struct  passwd *pwd;
    pwd = getpwuid(getuid());

    printf("\n%s@%s:%s $", pwd->pw_name, hostname, path);

    while (fgets(line, 256, stdin)) {
        arg_number = sscanf(line, "%s%s%s", args[0], args[1], args[2]);
        if (strcmp(args[0], "exit") == 0) {
            break;
        }
        else if (strcmp(args[0], "cd") == 0) {
            exec_cd();
        }
        else {
            exec();
        }

        char hostname[64];
        gethostname(hostname, sizeof(hostname));

        char path[128];
        getcwd(path, 128);

        struct  passwd *pwd;
        pwd = getpwuid(getuid());

        printf("\n%s@%s:%s $", pwd->pw_name, hostname, path);
    }

    return 0;
}
```

myshell.c