```c
/*
    COMP3511 Fall 2024
    PA1: Simplified Linux Shell (MyShell)

    Your name:LI,Yuntong
    Your ITSC email:ylino@connect.ust.hk

    Declaration:

    I declare that I am not involved in plagiarism
    I understand that both parties (i.e., students providing the codes and students copying the codes)

*/

/*
    Header files for MyShell
    Necessary header files are included.
    Do not include extra header files
*/
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h> // For constants that are required in open/read/write/close syscalls
#include <sys/wait.h> // For wait() - suppress warning messages
#include <fcntl.h>    // For open/read/write/close syscalls
#include <signal.h>   // For signal handling

// Define template strings so that they can be easily used in printf
//
// Usage: assume pid is the process ID
//
//  printf(TEMPLATE_MYSHELL_START, pid);
//
#define TEMPLATE_MYSHELL_START "Myshell (pid=%d) starts\n"
#define TEMPLATE_MYSHELL_END "Myshell (pid=%d) ends\n"
#define TEMPLATE_MYSHELL_TERMINATE "Myshell (pid=%d) terminates by Ctrl-C\n"

// Assume that each command line has at most 256 characters (including NULL)
#define MAX_CMDLINE_LENGTH 256

// Assume that we have at most 8 arguments
#define MAX_ARGUMENTS 8

// Assume that we only need to support 2 types of space characters:
// " " (space) and "\t" (tab)
#define SPACE_CHARS " \t"

// The pipe character
#define PIPE_CHAR "|"

// Assume that we only have at most 8 pipe segements,
// and each segment has at most 256 characters
#define MAX_PIPE_SEGMENTS 8

// Assume that we have at most 8 arguments for each segment
// We also need to add an extra NULL item to be used in execvp
// Thus: 8 + 1 = 9
//
// Example:
//   echo a1 a2 a3 a4 a5 a6 a7
//
// execvp system call needs to store an extra NULL to represent the end of the parameter list
//
//   char *arguments[MAX_ARGUMENTS_PER_SEGMENT];
//
//   strings stored in the array: echo a1 a2 a3 a4 a5 a6 a7 NULL
//
#define MAX_ARGUMENTS_PER_SEGMENT 9

// Define the standard file descriptor IDs here
```

```c
#define STDIN_FILENO 0  // Standard input
#define STDOUT_FILENO 1 // Standard output

// This function will be invoked by main()
// This function is given
int get_cmd_line(char *command_line)
{
    int i, n;
    if (!fgets(command_line, MAX_CMDLINE_LENGTH, stdin))
        return -1;
    // Ignore the newline character
    n = strlen(command_line);
    command_line[--n] = '\0';
    i = 0;
    while (i < n && command_line[i] == ' ')
    {
        ++i;
    }
    if (i == n)
    {
        // Empty command
        return -1;
    }
    return 0;
}

// read_tokens function is given
// This function helps you parse the command line
//
// Suppose the following variables are defined:
//
// char *pipe_segments[MAX_PIPE_SEGMENTS]; // character array buffer to store the pipe segements
// int num_pipe_segments; // an output integer to store the number of pipe segment parsed by this funct
// char command_line[MAX_CMDLINE_LENGTH]; // The input command line
//
// Sample usage:
//
//  read_tokens(pipe_segments, command_line, &num_pipe_segments, "|");
//
void read_tokens(char **argv, char *line, int *numTokens, char *delimiter)
{
    int argc = 0;
    char *token = strtok(line, delimiter);
    while (token != NULL)
    {
        argv[argc++] = token;
        token = strtok(NULL, delimiter);
    }
    *numTokens = argc;
}

void execute_command(char *cmd) {
    char *args[MAX_ARGUMENTS_PER_SEGMENT];
    int num_args;
    read_tokens(args, cmd, &num_args, SPACE_CHARS);
    args[num_args] = NULL;

    if (execvp(args[0], args) == -1) {
        perror("execvp");
        exit(EXIT_FAILURE);
    }
}

void sigint_handler(int sig) {
    printf(TEMPLATE_MYSHELL_TERMINATE, getpid());
    exit(0);
}

void process_cmd(char *command_line)
{
    // Uncomment this line to check the cmdline content
    // Please remember to remove this line before the submission
    // printf("Debug: The command line is [%s]\n", command_line);
```

```c
    char *pipe_segments[MAX_PIPE_SEGMENTS];
    int num_pipe_segments;
    read_tokens(pipe_segments, command_line, &num_pipe_segments, PIPE_CHAR);

    int pipe_fds[2 * (num_pipe_segments - 1)];

    for (int i = 0; i < num_pipe_segments - 1; i++) {
        if (pipe(pipe_fds + i * 2) < 0) {
            perror("pipe");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_pipe_segments; i++) {
        if (i != 0) {
            dup2(pipe_fds[(i - 1) * 2], STDIN_FILENO);
        }
        if (i != num_pipe_segments - 1) {
            dup2(pipe_fds[i * 2 + 1], STDOUT_FILENO);
        }
        for (int j = 0; j < 2 * (num_pipe_segments - 1); j++) {
            close(pipe_fds[j]);
        }

        execute_command(pipe_segments[i]);
        exit(0);
    }

    for (int i = 0; i < 2 * (num_pipe_segments - 1); i++) {
        close(pipe_fds[i]);
    }
    for (int i = 0; i < num_pipe_segments; i++) {
        wait(NULL);
    }
}
/* The main function implementation */
int main()
{
    // TODO: replace the shell prompt with your ITSC account name
    // For example, if you ITSC account is cspeter@connect.ust.hk
    // You should replace ITSC with cspeter
    char *prompt = "ylino";
    char command_line[MAX_CMDLINE_LENGTH];

    // TODO:
    // The main function needs to be modified
    // For example, you need to handle the exit command inside the main function
    signal(SIGINT, sigint_handler);
    printf(TEMPLATE_MYSHELL_START, getpid());

    // The main event loop
    while (1)
    {

        printf("%s> ", prompt);
        if (get_cmd_line(command_line) == -1)
            continue; /* empty line handling */
        if (strcmp(command_line, "exit") == 0) {
            printf(TEMPLATE_MYSHELL_END, getpid());
            break;
        }
        pid_t pid = fork();
        if (pid == 0)
        {
            //signal(SIGINT, SIG_DFL);
            // the child process handles the command
            process_cmd(command_line);
        }
        else
        {
            // the parent process simply wait for the child and do nothing
            wait(0);
        }
```

```
220:     }
221:
222:     return 0;
223: }
```