《操作系统原理》实验报告

实验名称	多进程编程	实验序号	2
实验日期	2023/3/27	实验人	盖乐

一、实验题目

设计一个 C 程序作为一个 shell 接口，它接受用户命令，然后在单独的进程中执行每个命令。在 shell 界面给用户一个提示，然后输入下一个命令。

要求：

1.单独的子进程是使用 fork() 系统调用创建的，用户的命令是使用系统调用 exec() 系列之一执行的

2.修改 shell 接口程序，使其提供 history 功能，允许用户访问最近输入的命令。通过使用该功能，用户最多可以访问 10 个命令。该程序还应该管理基本的错误处理。

二、相关原理与知识

为了分析用户输入的命令，我们可以使用函数 strtok()将其进行分割，以便找出要执行的命令和命令参数。同时，我们还需要比较输入的最后一个字符是否是“&”符号，以判断用户的命令是后台运行还是非后台运行，并将用户输入存储起来以便后续分析。

接下来，我们可以使用 fork()函数创建子进程。在父进程中，fork()函数返回的 pid 非 0，而在子进程中，返回的 pid 为 0。因此，我们可以根据 pid 的值来区分父进程和子进程，并实现不同的功能。

在子进程中，我们可以使用 exevp()函数来执行 shell 命令。而在父进程中，我们可以根据用户输入的最后一个字符是否为“&”来决定是否等待子进程结束，从而实现后台运行和非后台运行的功能。

三、实验过程

1.将一个字符串参数 param 按空格分隔成多个子字符串，并将它们存储到一个字符串数组 ret 中。该函数的返回值为一个整数，表示最后一个子字符串是否以 ″&″ 符号结尾。

```c
static int check_param(char* param, char* ret[max_param]) {
    int cnt = 0;
    char *p;
    strtok(param," ");
    ret[cnt++] = param;
    while((p = strtok(NULL, " "))) {
        ret[cnt++] = p;
    }
    if(ret[cnt-1][0] == '&') {
        ret[cnt-1] = NULL;
        return 1 ;
    }else{
        ret[cnt] = NULL;
        return 0;
    }
}
```

2.进行关键字定义，并在输入时进行检查，如果存在 exit、history 等关键词就进行相应操作。

```c
#define exit_flag "exit"
#define history_flag "history"
#define exclam_flag "!"
```

```c
static key_word check_key_word(char *key) {
    if(!strncmp(key, exit_flag, strlen(exit_flag)))
        return k_exit;
    if(! strncmp(key, history_flag, strlen(history_flag)))
        return k_history;
    if(!strncmp(key, exclam_flag, strlen(exclam_flag)))
        return k_exclam;
    return k_null;
}
```

3. 处理完特殊命令，用 fork、 execvp 执行新命令， 并将命令行入队列， 以供历史功能进行查询和使用。

```
    pid_t pid = fork();
    if(pid < 0) {
        fprintf(stderr, "%s\n", "ERROR: fork()");
        return ret;
    }else if(pid == 0) {
        if (back) {
            close(0);
            close(1);
            close(2);
        }
        execvp(param[0] ,param);
        perror("error");
        exit(0);
    }else {
        queue_add(obj, line);
        if (!back) {
            wait(&ret);
        }
        ret = 1;
    }
    return ret;
```

四、实验结果与分析

1. 执行命令，会打印出提示符 osh

```
gaile@gaile-virtual-machine:~/Code/operation system experiment$ ./ex2
osh>
```

2. 输入命令运行

```
osh>pwd
/home/gaile/Code/operation system experiment
osh>who
gaile    tty2         2023-03-29 21:40 (tty2)
osh>date
2023年 03月 29日 星期三 22:41:52 CST
```

3. 查看历史命令输入

```
osh>history
There are the latest 10 commands:
cmd[0]: date
cmd[1]: ls -ls
cmd[2]: ls -l
cmd[3]: ls -lh
cmd[4]: tade
cmd[5]: date
cmd[6]: cd
cmd[7]: dir
cmd[8]: ps
cmd[9]: ls -s
```

4. 当用户输入单!后跟整数 N，执行历史记录中的第 N 个命令

```
osh>!1
总用量 84
 4  1       4  cp       4  cp1.txt   4  op1.c
 4  2.c     0  cp1     20  ex2       4 'operation system .txt'
20  a.out   4  cp1.tx  16  op1
```

5. 当用户输入!!执行历史记录中的最新命令。

```
osh>ps
    PID TTY          TIME CMD
  48026 pts/0    00:00:00 bash
  62379 pts/0    00:00:00 ex2
  62680 pts/0    00:00:00 ps
osh>!!
    PID TTY          TIME CMD
  48026 pts/0    00:00:00 bash
  62379 pts/0    00:00:00 ex2
  62713 pts/0    00:00:00 ps
```

6. 如果历史中没有命令,输入!!应该会产生一条消息“No commands in history”。如果没有与用单个!输入的数字对应的命令，程序输出“No such command in history”进行基本的错误处理。

```
osh>!!
No command in history
osh>!1
No such command in history
```

7. 输入 exit 进行退出

```
osh>exit
gaile@gaile-virtual-machine:~/Code/operation system experiment$
```

五、 问题总结

1. strtok()函数直接在原字符串上分割， 使用前需要做好字符串备份。

```
char* param[max_param];
char line_brk[strlen(line)+1];
strcpy(line_brk, line);
```

2. 处于后台运行的子进程应该关闭 stdin stdout stderr， 保证 shell 上没有输出。

```
if (back) {
    close(0);
    close(1);
    close(2);
}
execvp(param[0] ,param);
perror("error");
exit(0);
```

六、源代码

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <memory.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <string.h>

#define max_length 80
#define max_param 16
#define max_history 10
#define prompt "osh>"
#define exit_flag "exit"
#define history_flag "history"
#define exclam_flag "!"

typedef enum _key_word {
```

```
    k_null, k_exit, k_history, k_exclam
}key_word;

typedef struct _cmd_queue {
    char* cmd[max_history];
    int rear;
}cmd_queue;

static int handle_input(char* line, cmd_queue* obj) {
    int ret = 1;
    char* param[max_param];
    char line_brk[strlen(line)+1];
    strcpy(line_brk, line);
    int back = check_param(line_brk, param);
    key_word key = check_key_word(param[0]);
    switch(key) {
        case k_exit:
            return 0;
        case k_history:
            do_history(obj);
            return 1;
        case k_exclam:
            return do_repeat(obj, param[0]);
        case k_null:
        default:
            break ;
    }
    pid_t pid = fork();
    if(pid < 0) {
        fprintf(stderr, "%s\n", "ERROR: fork()");
        return ret;
    }else if(pid == 0) {
```

```
            if (back) {
                close(0);
                close(1);
                close(2);
            }
            execvp(param[0] ,param);
            perror("error");
            exit(0);
        }else {
            queue_add(obj, line);
            if (!back) {
                wait(&ret);
            }
            ret = 1;
        }
        return ret;
}

static int check_param(char* param, char* ret[max_param]) {
        int cnt = 0;
        char *p;
        strtok(param," ");
        ret[cnt++] = param;
        while((p = strtok(NULL, " "))) {
            ret[cnt++] = p;
        }
        if(ret[cnt-1][0] == '&') {
            ret[cnt-1] = NULL;
            return 1 ;
        }else{
            ret[cnt] = NULL;
            return 0;
```

```
    }
}

static key_word check_key_word(char *key) {
    if(!strncmp(key, exit_flag, strlen(exit_flag)))
        return k_exit;
    if(! strncmp(key, history_flag, strlen(history_flag)))
        return k_history;
    if(!strncmp(key, exclam_flag, strlen(exclam_flag)))
        return k_exclam;
    return k_null;
}

static void do_history(cmd_queue* obj) {
    printf("There are the latest 10 commands: \n");
    queue_print(obj);
}

static int do_repeat(cmd_queue* obj,char* param) {
    int ret = 1;
    int pos = obj->rear;
    if(param[strlen(exclam_flag)] != '!')
        pos = atoi(param + strlen(exclam_flag));
    if(pos < 0 || pos > max_history || !(obj->cmd[pos])){
        if(!(obj->cmd[pos])){
            printf("No such command in history\n");
            return 1;
        }
        if(obj->cmd[0]==NULL){
            printf("No command in history\n");
        }
        return 1;
```

```
    }

    ret = handle_input(obj->cmd[pos], obj);
    return ret ;
}

static void queue_init(cmd_queue* obj) {
    if(obj == NULL)
        return;
    for(int i = 0; i < max_history; i++)
        obj->cmd[i] = NULL;
    obj->rear = -1;
}

static void queue_dele(cmd_queue* obj) {
    for(int i = 0;i<max_history; ++i){
        if(obj->cmd[i]) {
            free(obj->cmd[i]);
            obj->cmd[i] = NULL;
        }
    }
}

static void queue_add(cmd_queue* obj, char* cmd) {
    if(obj == NULL)
        return;
    int pos = (obj->rear++ + 1) % max_history;
    obj->cmd[pos] = (char* )realloc(obj->cmd[pos],
(1+strlen(cmd))*sizeof(char));
    strcpy (obj->cmd[pos], cmd);
}
```

```
static void queue_print(cmd_queue* obj) {
    if(obj->rear == -1)
        return ;
    int start = (obj->rear + 1) % max_history;
    while(!(obj->cmd[ start]))
        start = (start + 1) % max_history;

    for(int i = 0; i < max_history; ++i) {
        int pos = (start + i) % max_history;
        if( obj->cmd[pos])
            printf("cmd[%d]: %s\n", i, obj->cmd[pos]);
        else
            break ;
    }
}

int main()
{
    int run = 1;
    char buf[max_length];
    memset(buf, 0, max_length);
    cmd_queue history;
    queue_init(&history);
    while(run) {
        printf("%s", prompt);
        fflush(stdout);
        char* t = fgets(buf, max_length, stdin);
        if(t == NULL) {
            run = 0;
            break;
        }
        if(buf[strlen(buf)-1] == '\n')
```

```
            buf[strlen(buf)-1] = '\0';
        if(strlen(buf) <= 0)
            continue;
        run = handle_input(buf, &history);
    }
    queue_dele(&history);
    return 0;
}
```