|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **武汉大学国家网络安全学院教学实验报告** | | | | |
| 课程名称 | 操作系统及安全设计 | | 实验日期 | 2024/3/27 |
| 实验名称 | UNIX外壳和历史特征 | | 实验序号 |  |
| 姓 名 | 学 号 | 专业 | 年级-班 | 成绩 |
| 张杰 | 2022302181314 | 信安 | 22-08 |  |
| 陈佩桦 | 2022302181316 | 信安 | 22-09 |  |
| 张泽宇 | 2022302181313 | 网安 | 22-07 |  |
| 宣梓淅 | 2022302181310 | 网安 | 22-05 |  |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） | | | | |
| 1. 实验目的：设计一个C程序。这是一个外壳接口，可以接受用户命令，然后可以在一个单独进程中执行用户命令。这个项目可以在任何Linux、UNIX和Mac OSX系统上完成。外壳接口为用户提供提示符，以便输人下一个命令。 2. 实验内容：创建子进程并且在子进程中执行命令；修改外壳以便支持历史特征   3、原理分析：便分叉一个子进程并执行用户指定命令。这要求词法分析，以便分解用户输人为多个标记，并将这些标记存到字符串数组，这个数组args会被传递到函数execvp()，进而执行对应任务操作；用字符指针数组char\* line[MAXLINE]去存储每一条指令，并通过遍历去显示历史指令，通过数组编号和用户输入编号去寻找对应指令，并执行，若不在遍历范围或未输入过指令，则报错 | | | | |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） | | | | |
| 1. 实验环境：   本地机：Darwin jdeMacBook-Pro.local 23.2.0 Darwin Kernel Version 23.2.0: Wed Nov 15 21:53:18 PST 2023; root:xnu-10002.61.3~2/RELEASE\_ARM64\_T6000 arm64  虚拟机平台：parallels18  虚拟机：1. Linux euler 4.19.90 #1 SMP Thu Mar 14 03:04:50 CST 2024 aarch64 aarch64 aarch64 GNU/Linux   1. Linux localhost.localdomain 4.19.90 #3 SMP Sun Mar 17 22:16:12 CST 2024 x86\_64 x86\_64 x86\_64 GNU/Linux   2、实验步骤：  （因结果较长，截取部分页面展示出来）   1. 特殊功能定义（exit;history;!!;!n）    printf("412osh> ");          fflush(stdout);          // args[arg\_count] = (char\*)malloc(MAXLINE \* sizeof(char)); // 分配内存空间          // scanf("%s", args[arg\_count]);          // printf("%s\n",args[arg\_count]);          // 改用fgets()          line[line\_count] = (char\*)malloc(MAXLINE \* sizeof(char));          char tmpstr[MAXLINE];          fgets(tmpstr, MAXLINE, stdin);          strcpy(line[line\_count], tmpstr);          if (strcmp(line[line\_count], "exit\n") == 0)          {              should\_run = 0;              break;          }          else if (strcmp(line[line\_count], "history\n") == 0)          {              if (line\_count == 0)                  printf("NO commands history.\n");              else                  for (int i = line\_count - 1; i>=0; i-- )                  {                      printf("%d %s", i + 1, line[i]);                  }              continue;          }          else if (strcmp(line[line\_count], "!!\n")==0)          {              // printf("enter !!");              strcpy(tmpstr, line[line\_count-1]);              strcpy(line[line\_count], line[line\_count-1]);          }          else if ( tmpstr[0] == '!' && isdigit(tmpstr[1]) && tmpstr[2] == '\n')          {              // printf("enter !n \n");              int n;              n = tmpstr[1] - '0'; // 解析出整数 n              if(n>line\_count-1)              {                  printf("No such command in history.\n");                  continue;              }              else              {                  strcpy(tmpstr, line[n-1]);                  strcpy(line[line\_count], line[n - 1]);              }          }  2.词法分析，以便分解用户输人为多个标记，并将这些标记存到字符串数组args  char\* token = strtok(tmpstr, " \n\0");          int arg\_count = 0;          while (token != NULL && arg\_count < MAXLINE)          {              // 分配内存空间，并将单词存储在 args 数组中              args[arg\_count] = (char\*)malloc(strlen(token) + 1);              strcpy(args[arg\_count], token);              // 移动到下一个单词              token = strtok(NULL, " \n\0");              arg\_count++;          }          args[arg\_count] = NULL; // 最后一个参数设置为 NULL          line\_count++;          int should\_wait = 1;          if (arg\_count > 0 && strcmp(args[arg\_count - 1], "&") == 0) {              should\_wait = 0;              args[arg\_count - 1] = NULL; // 删除 & 符号          }  3.分叉一个子进程并执行用户指定命令          // 创建子进程          pid\_t pid = fork();          if (pid < 0) {              perror("fork");              exit(EXIT\_FAILURE);          }          else if (pid == 0) { // 子进程              // 执行用户指定的命令              execvp(args[0], args);              perror("execvp"); // 如果执行失败，则输出错误信息              exit(EXIT\_FAILURE); // 子进程退出          }          else { // 父进程              if (should\_wait) {                  // 等待子进程退出                  waitpid(pid, NULL, 0);              }          }  4.结果展示  Ubuntu 64 位-2024-03-27-15-23-47 | | | | |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） | | | | |
| 1. 实验分工：查资料；找教程学习；特殊功能定义；词法分析，将指令存入数组；分叉一个子进程执行命令   张杰：写程序  陈佩桦：写程序  宣梓淅：查资料  张泽宇：查资料    2、实验故障：一开始错认为字符指针数组返回地址而非字符串，导致词法分析时的字符串有误；对execvp传入的参数理解有误，未意识到作为指令的参数最后需设置为NULL （arg[arg\_count] = NULL）以及 识别到“&”符后忘记删除； | | | | |
| 1. 实验结果总结   （对实验结果进行分析，完成思考题目，总结实验成员每人的实验心得体会，并提出实验的改进意见）  本次大作业的实质是让我们实现一个简易的shell。能执行用户指定的命令，也能识别特殊指令完成操作，当然对于父子进程的使用和父子进程之间的关系也有了清晰的认知。我们小组分工合作，厘清了实现一个shell需要的一些基本步骤，随后高效的完成了任务。此次我们在实现shell的过程当中也顺便搞清楚了shell和终端的密切相关的关系。 | | | | |
| 1. 教师评语 | | | | |
| 全组评语： | | | | |
|  | | | | |
| 个人评语（请填写本组同学姓名，队长排第一，其他同学随后） | | | | |
| 1. 张杰：认识了一个神奇的函数execvp，和它的奇怪的参数。主要的工作就是用c语言写个shell以及基本的一些词法分析很简单，没什么难度。 2. 陈佩桦：对于我来说，本次实验的主要收获是知道了父子进程的关系，如waitpid()实现等待子进程退出；实现shell的一些基本步骤（传入指令、分割单词、词法分析，调用函数执行命令）。本次实验也让我再次搞清楚了终端和shell的关系，终端提供用户界面，shell负责解释用户输入的命令。总的来说，对于外壳接口有了深入的理解。 3. 张泽宇：搞清楚了外壳接口实现的基本原理和步骤，也知道了父子进程之间的先后关系，具体实现过程正常。属于是对shell的基本了解并操作。 4. 宣梓淅：在查找资料了解shell的时候，对于外壳的原理与作用有了更加清晰深入的了解。理解了不同函数的用法，以及如何接受命令并执行。 | | | | |
| 教师签名： | | | | |