**《操作系统原理》**

**实 验 报 告 书**

**学 生 姓 名**

**学 号**

**班 级**

**2021 — 2022 学年 第 2 学期**

|  |  |  |  |
| --- | --- | --- | --- |
| **《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** | **2022/6** | **实验人** |  |
| **一**        **这个项目设计一个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，外壳接口为用户提供提示符。** | | | |
| 1. **相关原理与知识**   **1.Shell**  **Shell原意为“壳”，它包裹在内核之外，处于用户与内核之间。其最主要的功能为接收用户输入的命令，找到命令所在的位置，并加以执行。实现shell接口的一种技术是让父进程首先读取用户输入的命令，然后创建一个单独的子进程完成该命令。Unix风格的shell都内置有历史记录功能**  **2.fork（）函数：**  **一个进程，包括代码、数据和分配给进程的资源。fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。**  **一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。**  **3.exec（）函数：**  **用fork函数创建子进程后，子进程往往要调用一种exec函数以执行另一个程序。当进程调用一种exec函数时，该进程执行的程序完全替换为新程序，而新程序则从其main函数开始执行。因为调用exec并不创建新进程，所以前后的进程ID并未改变。exec只是用一个全新的程序替换了当前进程的正文、数据、堆和栈段。**  **4.wait（）**  **一个进程在终止时会关闭所有文件描述符，释放在用户空间释放的内存，但它**  **的PCB还保留着，内核在其中保存一些信息：如果是正常终止时则保存着退出状态，如果是异常终止则保存着导致该进程终止的信号是哪个，这个进程的父进程可以调用wait或waitpid获取这些信息，然后彻底清除这个进程。** | | | |
| 1. **实验过程**   **1.设计c程序作为shell接口，能够接受用户命令，然后在一个单独的进程中设计执行每个命令。**  **2.使用gcc进行程序编译和运行，测试代码功能。**  **3.对shell程序进行修改，使其能够提供历史特征，允许用户成功执行最近输入的命令。最多访问10条命令，并且该程序能够管理基本的错误处理。**  **4.再次使用gcc进行程序编译和运行，测试代码功能。** | | | |
| 1. **实验结果与分析**   **1.** **执行该程序，实现外壳接口,编写c程序并运行：**      **2.外壳接口程序能够列出命令历史：**  **通过在提示符后输入history，将会输出最近执行的几条命令。**  **若输入！！时，将会执行最近的历史命令；**  **若输入！n时，将会执行第n个历史命令。**      1.创建子进程  （1）将用户输入的命令保存在一个数组args[]内。  定义一个函数int read\_command(char \*\*args);  实现分解用户输入为多个标记，并将这些标记存到字符串数组中。最后函数返回值为命令词数command\_count。  （2）将数组中的内容传递给execvp()函数中执行。  定义一个函数int command(char \*\*args,int command\_count)  实现分叉一个子进程并执行用户指定命令。  参数数组arg[]和command\_count传入该函数execvp()中执行数组内的命令。  检查用户输入是否包括一个&，以便确定父进程是否等待子进程退出。  2创建历史功能  定义一个函数int write\_history(char \*tmp)保存用户历史输入过的命令，存放在一个数组中history[]中；定义一个int (char \*\*args,int n)函数读出数组arg[]中存放过的命令。程序中包含各种错误输入提醒。最后在main()函数中，实现当用户输入history命令时，依次循环输出数组history中的命令，当用户输入！！时，执行read\_history\_command函数，读取最近的一个用户命令并执行，当用户输入！N时，执行第n条历史命令。 | | | |
| **五、问题总结**  **要熟悉掌握Linux系统常用进程创建与管理的系统调用，linux 下使用fork()创建子进程，通过实验更好的理解和掌握了进程的创建，对于进程的管理的理解也有了清晰地认识。** | | | |
| **六、源代码**  **#include<stdio.h>**  **#include <unistd.h>**  **#include<string.h>**  **#include<stdlib.h>**  **#include <wait.h>**  **#define MAX\_LINE 80**  **int input(char\* str);**  **int str\_to\_args(char\* str,char\* args[],int str\_num);**  **void display\_history(int history\_front,int history\_rear);**  **void add\_history(char\* str,int\* front,int\* rear);**  **char history[10][MAX\_LINE];//history queue**  **int input(char\* str){**  **char c;**  **int i = 0;**  **while((c = getchar())!='\n' && i<MAX\_LINE){**  **str[i] = c;**  **i++;**  **}**  **if(i == MAX\_LINE && c != '\n'){//when error**  **printf("over maximum length!");**  **return 0;**  **}**  **else{**  **str[i] = 0;**  **return i;**  **}**  **}**  **int str\_to\_args(char\* str,char\* args[],int str\_num){**  **const char s[2] = " ";**  **int i = 0;**  **char\* temp;**  **temp = strtok(str,s);**  **while(temp != NULL){**  **args[i] = (char\*)malloc(strlen(temp));**  **strcpy(args[i],temp);**  **i++;**  **temp = strtok(NULL,s);**  **}**  **args[i] = 0;**  **return i;**  **}**  **void display\_history(int history\_front,int history\_rear){**  **int i;**  **for(i = history\_front;i < history\_rear;i++){**  **printf("%d\t%s\n",history\_rear - i,history[i%10]);**  **}**  **}**  **void add\_history(char\* str,int\* front,int\* rear){**  **strcpy(history[\*rear % 10],str);//add**  **\*rear = \*rear+1;**  **if(\*rear - \*front > 10)**  **\*front++;**  **}**  **int main(void){**  **char\* args[MAX\_LINE/2+1];//execvp's argslist**  **int should\_run = 1;**  **int args\_num = 0;**  **char str[MAX\_LINE];// user input**  **int str\_num;**  **int history\_front = 0;**  **int history\_rear = 0;**    **int i = 0;**  **int background = 0;**  **int pid\_status;**  **while(should\_run){**  **printf("osh>");**  **fflush(stdout);**    **str\_num = input(str);//get user input**  **if(str\_num == 0){//no input**  **continue;**  **}**  **if(strcmp(str,"exit")== 0){//exit**  **should\_run = 0;**  **continue;**  **}**  **if(strcmp(str,"history") == 0){//display history**  **display\_history(history\_front,history\_rear);**  **}**  **if(strcmp(str,"!!") == 0){//use recent command**  **if(history\_rear != 0){**  **strcpy(str,history[(history\_rear -1) % 10]);**  **str\_num = strlen(str);**  **}**  **else{**  **printf("No commands in history.\n");**  **}**  **}**  **else if(str[0] == '!'){//use history command**  **if(str[1] <= '0' || str[1] > '9' || (str[1]-'0') > history\_rear-history\_front){**  **printf("No such command in history.\n");**  **}**  **else if(str[1] == '1'){**  **if(str[2] == '0')**  **strcpy(str,history[history\_front %10]);**  **else if(str[2] == 0)**  **strcpy(str,history[(history\_rear-1) %10]);**  **else**  **printf("No such command in history.\n");**  **}**  **else**  **strcpy(str,history[history\_rear - str[1] + '0' -1]);**  **}**  **add\_history(str,&history\_front,&history\_rear);//add to history**  **args\_num = str\_to\_args(str,args,str\_num);**  **if(strcmp(args[args\_num-1],"&") == 0){**  **background = 1;**  **args\_num--;**  **args[args\_num] = NULL;**  **}**  **pid\_t pid = fork();**  **if(pid == 0){**  **pid\_status = execvp(args[0], args);**  **}**  **else{**  **if(background == 1){**  **printf("%d is running in background %s \n",pid,str);**  **}**  **else{**  **wait(&pid\_status);**  **}**  **}**  **background = 0;**  **}**  **return 0;**  **}** | | | |