**Lab2\_添加Linux系统调用**

**PB19051183 吴承泽**

**一、编写系统调用实现一个Linux ps**

**1设计思路：**

**首先，先注册一个系统调用，由于上一个注册的是332，这一个注册的设为333。在syscalls.h中定义函数原型中，\*num指的是进程个数，\*\*command是二维指针，负责传出进程名字的char型数组，\*pid为一维整型数组，\*stime为一维整形数组，分别用来将内核中的数据传出。**

**在测试代码中，我设计了三个数组，分别通过syscall(333, &result,command,pid,stime);来传出进程个数，进程名字，pid和stime。再将其全部打印出来即可。**

**2代码部分：**

**写入syscall\_64.btl的代码：**

**333 common process\_status sys\_process\_status**

**写入syscalls.h的代码：**

**asmlinkage long sys\_process\_status(int \_\_user \* num, char \_\_user \*\* command, int \_\_user \* pid, int \_\_user \* cputime);**

**写入sys.c的代码：**

**SYSCALL\_DEFINE4(process\_status, int \_\_user \*, num, char \_\_user \*\*, command, int \_\_user \*, pid, int \_\_user \*, stime)**

**{**

**struct task\_struct\* task;**

**int counter = 0;**

**int pid\_temp[100];**

**int stime\_temp[100];**

**char command\_temp[100][16];**

**int i = 0;**

**printk("[Syscall] process\_status\n");**

**for\_each\_process(task)**

**{**

**i = 0;**

**while(i < 16)**

**{**

**command\_temp[counter][i] = task->comm[i];**

**i++;**

**}**

**pid\_temp[counter] = task->pid;**

**stime\_temp[counter] = task->stime;**

**counter ++;**

**}**

**copy\_to\_user(command, command\_temp, 100 \* 16 \* sizeof(char));**

**copy\_to\_user(pid, pid\_temp,100 \* sizeof(int));**

**copy\_to\_user(stime, stime\_temp,100 \* sizeof(int));**

**copy\_to\_user(num, &counter, sizeof(int));**

**return 0;**

**}**

**3.测试代码：**

**#include<linux/unistd.h>**

**#include<sys/syscall.h>**

**#include<stdio.h>**

**#include<unistd.h>**

**#define MAX 100**

**int main(void)**

**{**

**int pid[MAX];**

**char command[MAX][16];**

**int stime[MAX];**

**int result;**

**syscall(333, &result,command,pid,stime);**

**printf("process number is %d\n",result);**

**printf("PID\tTIME/ms\tCOMMAND\n");**

**for(int i = 0; i < result; i++)**

**{**

**printf("%d\t%d\t%s\n", pid[i],stime[i],command[i]);**

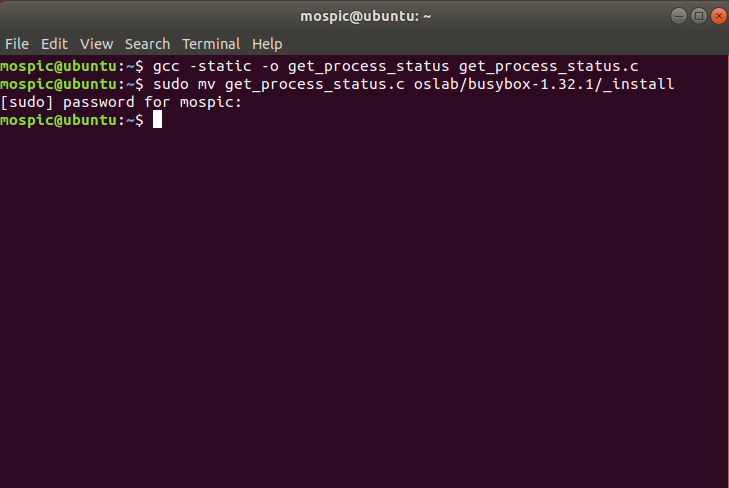
**}**

**return 0;**

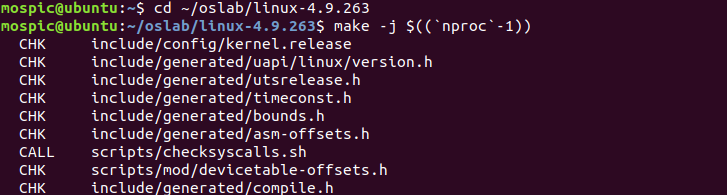
**}**

**3.实现过程：**

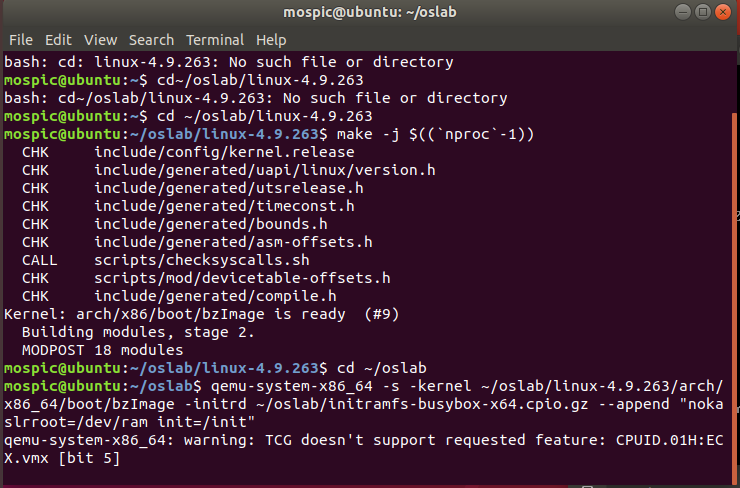
**1.编译测试代码**



**2.运行测试程序**

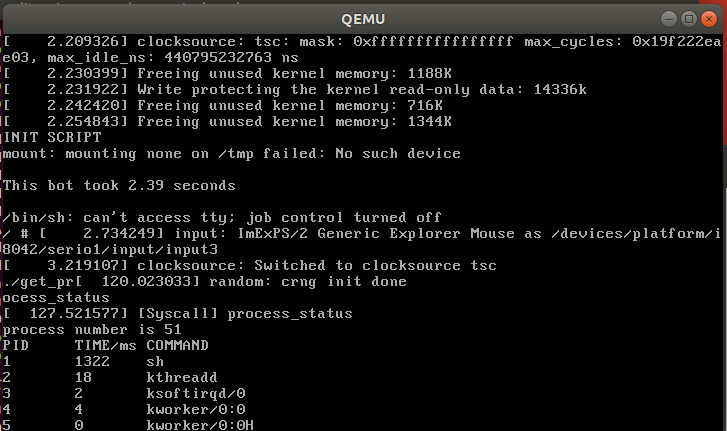


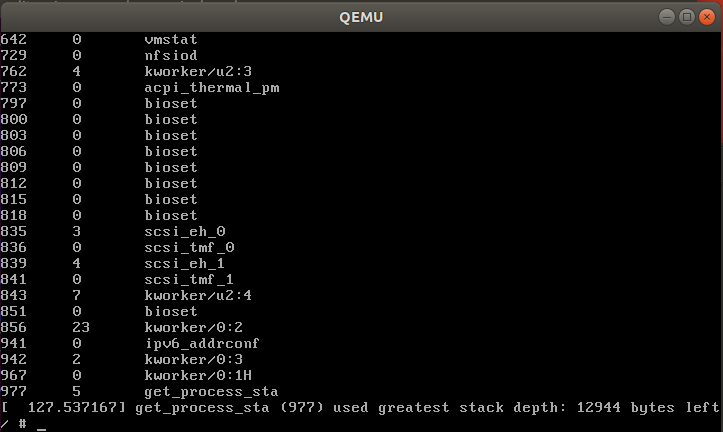
**3.执行QEMU**



**4.实现在QEMU运行类似ps指令功能**

**输入./get\_process\_status 输出process的数量 PID 时间 进程名**





**二、实现一个Linux Shell**

**1.设计思路**

**设计一个自己的linux shell，在要求中主要分为三部分，第一部分为：当输入中没有管道时， 若不是exit或cd指令，则fork出一个子程序，执行该程序。第二部分为：当输入中存在一个管道时，则生成一个管道，将管道与父进程和子进程相连，在子进程中执行第一个程序，将标准输出写入管道中，再fork出一个新子进程，将上一个子进程的输出当做标准输入通过管道放进该进程中，执行，结束，关闭管道。第三部分为：当输入中存在多个管道时，每次执行时生成一个管道，每次执行的时候fork出一个子程序，通过read\_fd标识符，将上一个子程序的输出放进管道中当做下一个子程序的标准输入，因此通过多次交换数据，可以实现多个管道的命令。**

**2.代码部分**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/types.h>

#define MAX\_CMDLINE\_LENGTH 1024 /\* max cmdline length in a line\*/

#define MAX\_BUF\_SIZE 4096 /\* max buffer size \*/

#define MAX\_CMD\_ARG\_NUM 32 /\* max number of single command args \*/

#define WRITE\_END 1 // pipe write end

#define READ\_END 0 // pipe read end

/\*

\* 闇€瑕佸ぇ瀹跺畬鎴愮殑浠ｇ爜宸茬粡鐢ㄦ敞閲奰TODO:`鏍囪

\* 鍙互缂栬緫鍣ㄦ悳绱㈡壘鍒?

\* 浣跨敤鏀寔TODO楂樹寒缂栬緫鍣紙濡倂scode瑁匱ODO highlight鎻掍欢锛夌殑鍚屽鍙互杞绘澗鎵惧埌瑕佹坊鍔犲唴瀹圭殑鍦版柟銆?

\*/

/\*

int split\_string(char\* string, char \*sep, char\*\* string\_clips);

function: split string by sep and delete whitespace at clips' head & tail

arguments: char\* string, Input, string to be splitted

char\* sep, Input, the symbol used to split string

char\*\* string\_clips, Input/Output, giving an allocated char\* array

and return the splitted string array

return value: int, number of splitted strings

\*/

int split\_string**(**char**\*** string**,** char **\***sep**,** char**\*\*** string\_clips**)** **{**

char string\_dup**[**MAX\_BUF\_SIZE**];**

string\_clips**[**0**]** **=** strtok**(**string**,** sep**);**

int clip\_num**=**0**;**

**do** **{**

char **\***head**,** **\***tail**;**

head **=** string\_clips**[**clip\_num**];**

tail **=** head **+** strlen**(**string\_clips**[**clip\_num**])** **-** 1**;**

**while(\***head **==** ' ' **&&** head **!=** tail**)**

head **++;**

**while(\***tail **==** ' ' **&&** tail **!=** head**)**

tail **--;**

**\*(**tail **+** 1**)** **=** '\0'**;**

string\_clips**[**clip\_num**]** **=** head**;**

clip\_num **++;**

**}while(**string\_clips**[**clip\_num**]=**strtok**(NULL,** sep**));**

**return** clip\_num**;**

**}**

/\*

鎵ц鍐呯疆鍛戒护

arguments:

argc: 鍛戒护鐨勫弬鏁颁釜鏁?

argv: 渚濇浠ｈ〃姣忎釜鍙傛暟锛屾敞鎰忕涓€涓弬鏁板氨鏄鎵ц鐨勫懡浠わ紝

鑻ユ墽琛?ls a b c"鍛戒护锛屽垯argc=4, argv={"ls", "a", "b", "c"}

return:

int, 鑻ユ墽琛屾垚鍔熻繑鍥?锛屽惁鍒欒繑鍥炲€奸潪闆?

\*/

int exec\_builtin**(**int argc**,** char**\*\***argv**)** **{**

**if(**argc **==** 0**)** **{**

**return** 0**;**

**}**

/\* TODO: 娣诲姞鍜屽疄鐜板唴缃寚浠?\*/

**if** **(**strcmp**(**argv**[**0**],** "cd"**)** **==** 0**)**

**{**

chdir**(**argv**[**1**]);**

**return** 0**;**

**}**

**else** **if** **(**strcmp**(**argv**[**0**],** "pwd"**)** **==** 0**)**

**{**

//char \*buffer = (char \*)malloc(MAX\_BUF\_SIZE \* (sizeof(char)));

//getcwd(buffer,MAX\_BUF\_SIZE);

//printf("%s\n",buffer);

//free(buffer);

**return** 0**;**

**}** **else** **if** **(**strcmp**(**argv**[**0**],** "exit"**)** **==** 0**)**

**{**

//if(argv[1] != '\0')

//return (argv[0] - '0');

//else

//return 0;

exit**(**0**);**

**}** **else** **{**

// 涓嶆槸鍐呯疆鎸囦护鏃?

**return** **-**1**;**

**}**

**}**

/\*

鍦ㄦ湰杩涚▼涓墽琛岋紝涓旀墽琛屽畬姣曞悗缁撴潫杩涚▼銆?

arguments:

argc: 鍛戒护鐨勫弬鏁颁釜鏁?

argv: 渚濇浠ｈ〃姣忎釜鍙傛暟锛屾敞鎰忕涓€涓弬鏁板氨鏄鎵ц鐨勫懡浠わ紝

鑻ユ墽琛?ls a b c"鍛戒护锛屽垯argc=4, argv={"ls", "a", "b", "c"}

return:

int, 鑻ユ墽琛屾垚鍔熷垯涓嶄細杩斿洖锛堣繘绋嬬洿鎺ョ粨鏉燂級锛屽惁鍒欒繑鍥為潪闆?

\*/

int execute**(**int argc**,** char**\*\*** argv**)** **{**

**if(**exec\_builtin**(**argc**,** argv**)** **==** 0**)** **{**

exit**(**0**);**

**}**

/\* TODO:杩愯鍛戒护 \*/

char buffer**[**MAX\_BUF\_SIZE**]** **=** "/bin/"**;**

strcat**(**buffer**,**argv**[**0**]);**

execl**(**buffer**,**argv**[**0**],**argv**[**1**],NULL);**

exit**(**0**);**

**}**

int main**()** **{**

/\* 杈撳叆鐨勫懡浠よ \*/

char cmdline**[**MAX\_CMDLINE\_LENGTH**];**

/\* 鐢辩閬撴搷浣滅'|'鍒嗗壊鐨勫懡浠よ鍚勪釜閮ㄥ垎锛屾瘡涓儴鍒嗘槸涓€鏉″懡浠?\*/

char **\***commands**[**128**];**

int cmd\_count**;**

**while** **(**1**)** **{**

/\* TODO:澧炲姞鎵撳嵃褰撳墠鐩綍锛屾牸寮忕被浼?shell:/home/oslab ->"锛屼綘闇€瑕佹敼涓嬮潰鐨刾rintf \*/

char **\***buffer\_temp **=** **(**char **\*)**malloc**(**MAX\_BUF\_SIZE **\*** **(sizeof(**char**)));**

getcwd**(**buffer\_temp**,**MAX\_BUF\_SIZE **\*** **sizeof(**char**));**

printf**(**"shell: %s -> "**,** buffer\_temp**);**

fflush**(**stdout**);**

free**(**buffer\_temp**);**

fgets**(**cmdline**,** 256**,** stdin**);**

strtok**(**cmdline**,** "\n"**);**

/\* 鎷嗚В鍛戒护琛?\*/

cmd\_count **=** split\_string**(**cmdline**,** "|"**,** commands**);**

**if(**cmd\_count **==** 0**)** **{**

**continue;**

**}** **else** **if(**cmd\_count **==** 1**)** **{** // 娌℃湁绠￠亾鐨勫崟涓€鍛戒护

char **\***argv**[**MAX\_CMD\_ARG\_NUM**];**

int argc **=** split\_string**(**commands**[**0**],** " "**,** argv**);**

/\* TODO:澶勭悊鍙傛暟锛屽垎鍑哄懡浠ゅ悕鍜屽弬鏁?

\*

\*

\*

\*/

/\* 鍦ㄦ病鏈夌閬撴椂锛屽唴寤哄懡浠ょ洿鎺ュ湪涓昏繘绋嬩腑瀹屾垚锛屽閮ㄥ懡浠ら€氳繃鍒涘缓瀛愯繘绋嬪畬鎴?\*/

**if(**exec\_builtin**(**argc**,** argv**)** **==** 0**)** **{**

**continue;**

**}**

int pid **=** fork**();**

**if(**pid **<** 0**)**

**{**

fprintf**(**stderr**,** "Fork Failed"**);**//fprintf

// return 1;

**}**

**else** **if(**pid **==** 0**)**

**{**

execute**(**argc**,** argv**);**

**}**

wait**(NULL);**

/\* TODO:鍒涘缓瀛愯繘绋嬶紝杩愯鍛戒护锛岀瓑寰呭懡浠よ繍琛岀粨鏉?

\*

\*

\*

\*

\*/

**}** **else** **if(**cmd\_count **==** 2**)** **{** // 涓や釜鍛戒护闂寸殑绠￠亾

int pipefd**[**2**];**

int ret **=** pipe**(**pipefd**);**

**if(**ret **<** 0**)** **{**

printf**(**"pipe error!\n"**);**

**continue;**

**}**

// 瀛愯繘绋?

int pid **=** fork**();**

**if(**pid **==** 0**)** **{**

/\*TODO:瀛愯繘绋? 灏嗘爣鍑嗚緭鍑洪噸瀹氬悜鍒扮閬擄紝娉ㄦ剰杩欓噷鏁扮粍鐨勪笅鏍囪鎸栫┖浜嗚琛ュ叏\*/

close**(**pipefd**[**0**]);**

dup2**(**pipefd**[**1**],** STDOUT\_FILENO**);**

close**(**pipefd**[**1**]);**

/\*

鍦ㄤ娇鐢ㄧ閬撴椂锛屼负浜嗗彲浠ュ苟鍙戣繍琛岋紝鎵€浠ュ唴寤哄懡浠や篃鍦ㄥ瓙杩涚▼涓繍琛?

鍥犳鎴戜滑鐢ㄤ簡涓€涓皝瑁呭ソ鐨別xecute鍑芥暟

\*/

char **\***argv**[**MAX\_CMD\_ARG\_NUM**];**

int argc **=** split\_string**(**commands**[**0**],** " "**,** argv**);**

**if(**exec\_builtin**(**argc**,** argv**)** **==** 0**)** **{**

**continue;**

**}**

execute**(**argc**,** argv**);**

exit**(**255**);**

**}**

// 鍥犱负鍦╯hell鐨勮璁′腑锛岀閬撴槸骞跺彂鎵ц鐨勶紝鎵€浠ユ垜浠笉鍦ㄦ瘡涓瓙杩涚▼缁撴潫鍚庢墠杩愯涓嬩竴涓?

// 鑰屾槸鐩存帴鍒涘缓涓嬩竴涓瓙杩涚▼

// 瀛愯繘绋?

pid **=** fork**();**

**if(**pid **==** 0**)** **{**

/\* TODO:瀛愯繘绋? 灏嗘爣鍑嗚緭鍏ラ噸瀹氬悜鍒扮閬擄紝娉ㄦ剰杩欓噷鏁扮粍鐨勪笅鏍囪鎸栫┖浜嗚琛ュ叏 \*/

close**(**pipefd**[**1**]);**

dup2**(**pipefd**[**0**],** STDIN\_FILENO**);**

close**(**pipefd**[**0**]);**

char **\***argv**[**MAX\_CMD\_ARG\_NUM**];**

int argc **=** split\_string**(**commands**[**1**],** " "**,** argv**);**

**if(**exec\_builtin**(**argc**,** argv**)** **==** 0**)** **{**

**continue;**

**}**

execute**(**argc**,** argv**);**

exit**(**255**);**

/\* TODO:澶勭悊鍙傛暟锛屽垎鍑哄懡浠ゅ悕鍜屽弬鏁帮紝骞朵娇鐢╡xecute杩愯

\* 鍦ㄤ娇鐢ㄧ閬撴椂锛屼负浜嗗彲浠ュ苟鍙戣繍琛岋紝鎵€浠ュ唴寤哄懡浠や篃鍦ㄥ瓙杩涚▼涓繍琛?

\* 鍥犳鎴戜滑鐢ㄤ簡涓€涓皝瑁呭ソ鐨別xecute鍑芥暟

\*

\*

\*/

**}**

close**(**pipefd**[**WRITE\_END**]);**

close**(**pipefd**[**READ\_END**]);**

**while** **(**wait**(NULL)** **>** 0**);**

**}** **else** **{** // 涓変釜浠ヤ笂鐨勫懡浠?

int read\_fd**;** // 涓婁竴涓閬撶殑璇荤鍙ｏ紙鍑哄彛锛?

**for(**int i**=**0**;** i**<**cmd\_count**;** i**++)** **{**

int pipefd**[**2**];**

**if(**i **!=** cmd\_count **-** 1**)**

**{**

int ret **=** pipe**(**pipefd**);**

**if(**ret **<** 0**)**

**{**

printf**(**"pipe error!\n"**);**

**continue;**

**}**

**}**

/\* TODO:鍒涘缓绠￠亾锛宯鏉″懡浠ゅ彧闇€瑕乶-1涓閬擄紝鎵€浠ユ湁涓€娆″惊鐜腑鏄笉鐢ㄥ垱寤虹閬撶殑

\*

\*

\*

\*/

int pid **=** fork**();**

**if(**pid **==** 0**)** **{**

**if(**i **!=** cmd\_count **-** 1**)**

**{**

close**(**pipefd**[**0**]);**

dup2**(**pipefd**[**1**],** STDOUT\_FILENO**);**

close**(**pipefd**[**1**]);**

**}**

/\* TODO:闄や簡鏈€鍚庝竴鏉″懡浠ゅ锛岄兘灏嗘爣鍑嗚緭鍑洪噸瀹氬悜鍒板綋鍓嶇閬撳叆鍙?

\*

\*

\*

\*/

**if(**i **!=** 0**)**

**{**

dup2**(**read\_fd**,** STDIN\_FILENO**);**//

close**(**read\_fd**);**

**}**

/\* TODO:闄や簡绗竴鏉″懡浠ゅ锛岄兘灏嗘爣鍑嗚緭鍑洪噸瀹氬悜鍒颁笂涓€涓閬撳叆鍙?

\*

\*

\*

\*/

char **\***argv**[**MAX\_CMD\_ARG\_NUM**];**

int argc **=** split\_string**(**commands**[**i**],** " "**,** argv**);**

**if(**exec\_builtin**(**argc**,** argv**)** **==** 0**)** **{**

**continue;**

**}**

execute**(**argc**,** argv**);**

exit**(**255**);**

/\* TODO:澶勭悊鍙傛暟锛屽垎鍑哄懡浠ゅ悕鍜屽弬鏁帮紝骞朵娇鐢╡xecute杩愯

\* 鍦ㄤ娇鐢ㄧ閬撴椂锛屼负浜嗗彲浠ュ苟鍙戣繍琛岋紝鎵€浠ュ唴寤哄懡浠や篃鍦ㄥ瓙杩涚▼涓繍琛?

\* 鍥犳鎴戜滑鐢ㄤ簡涓€涓皝瑁呭ソ鐨別xecute鍑芥暟

\*

\*

\*/

**}**

/\* 鐖惰繘绋嬮櫎浜嗙涓€鏉″懡浠わ紝閮介渶瑕佸叧闂綋鍓嶅懡浠ょ敤瀹岀殑涓婁竴涓閬撹绔彛

\* 鐖惰繘绋嬮櫎浜嗘渶鍚庝竴鏉″懡浠わ紝閮介渶瑕佷繚瀛樺綋鍓嶅懡浠ょ殑绠￠亾璇荤鍙?

\* 璁板緱鍏抽棴鐖惰繘绋嬫病鐢ㄧ殑绠￠亾鍐欑鍙?

\*

\*/

// 鍥犱负鍦╯hell鐨勮璁′腑锛岀閬撴槸骞跺彂鎵ц鐨勶紝鎵€浠ユ垜浠笉鍦ㄦ瘡涓瓙杩涚▼缁撴潫鍚庢墠杩愯涓嬩竴涓?

// 鑰屾槸鐩存帴鍒涘缓涓嬩竴涓瓙杩涚▼

**if(**i **!=** 0**)**

close**(**read\_fd**);**

**if(**i **!=** cmd\_count **-** 1**)**

read\_fd **=** pipefd**[**0**];**

close**(**pipefd**[**1**]);**

**}**

**while** **(**wait**(NULL)** **>** 0**);**

// TODO:绛夊緟鎵€鏈夊瓙杩涚▼缁撴潫

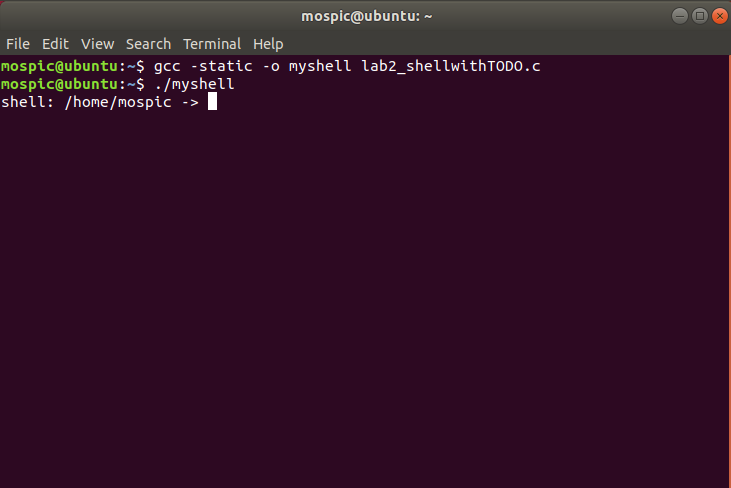
**}**

**}**

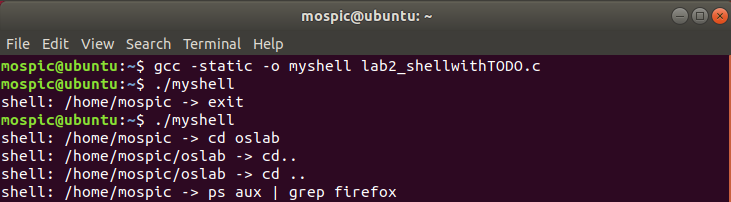
**}**

**3.实现过程**

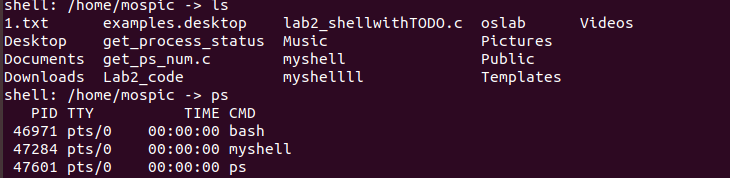
**编译shell的C文件：**



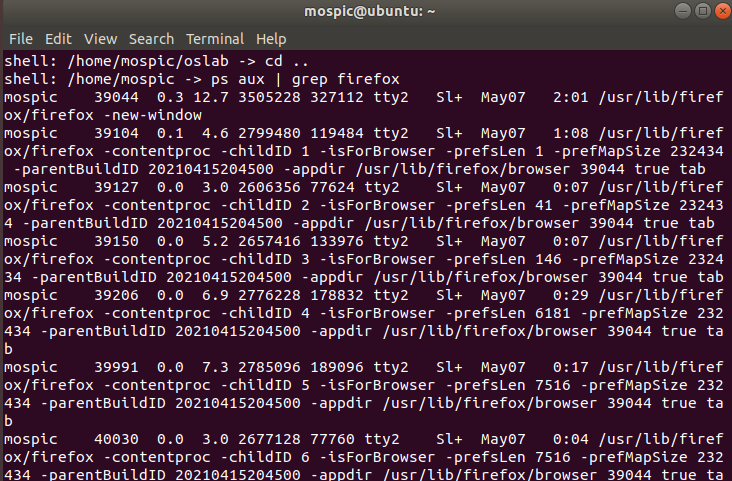
**内置指令的实现：**

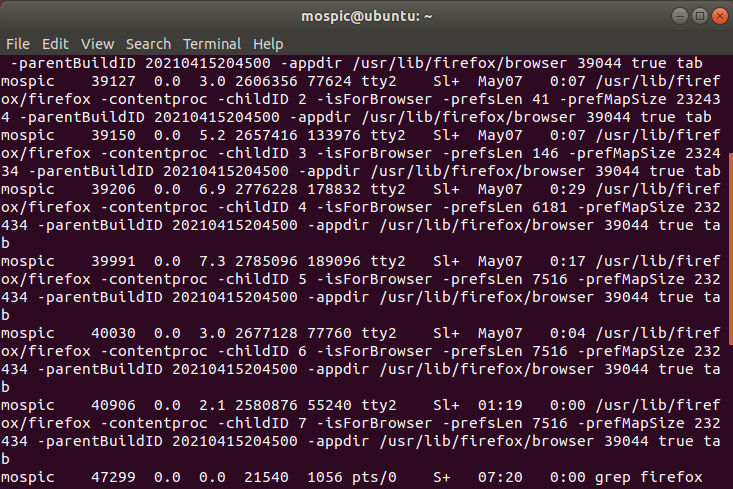


**外置指令的实现：**

****

**双管道shell实现：**





**多管道shell实现：**

