Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №6

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Студенты: Суслов А.В.

Эльвейн Н.Л.  
 Олимпиади М.Б.

Преподаватели: Хайленко Е.А.  
 Филиппова Е.В.

Вариант: 1

1. **Цель работы**

Практическое освоение средств управления ресурсами ОС UNIX на основе

разработки программы, моделирующей работу интерпретатора в плане создания

процессов, реализующих команды в командной строке, их синхронизации и

взаимодействию по данным.

1. **Задание**

Составить программу, моделирующую работу Shell-интерпретатора при

обработке командной строки, указанной в варианте. При реализации программы

путем выдачи сообщений информировать обо всех этапах ее работы (создан

процесс, выполнение команды закончено и т.д.).

Вариант №1:

(cc pr1.c & cc pr2.c) && cat pr1.c pr2.c > prall.c

1. **Описание использованных системных вызовов**
   1. **Функции работы с файлами/каналами**

**int dup2(int \_\_fd, int \_\_fd2) <unistd.h>**

Дублирует дескриптор, заданный первым параметром, чтобы он указывал на \_\_fd2, перед этим закрывая \_\_fd2, чтобы они указывали на один и тот же файл.

**int open(const char \*\_\_path, int \_\_oflag, ...) <fcntl2.h>**

Открывает файл, возвращая дескриптор, указывающий на него (а не потоковый тип, как в случае с fopen()). Значение третьего параметра варьируется в зависимости от выставленных во втором параметре флагов. Возможные значения флагов: O\_RDONLY, O\_WRONLY, O\_CREAT, тысячи их. Флаги можно комбинировать с использованием логической операции **|** (OR)

**int fcntl (int \_\_fd, int \_\_cmd, ...) <fcntl.h>**

Совершает указанную в \_\_cmd операцию над файловым дескриптором \_\_fd. Значения необязательных аргументов зависят от выбранной операции.

* 1. **Функции работы с процессами**

**pid\_t wait(void \*\_\_stat\_loc) <sys/wait.h>**

Ожидает поступления сигнала SIGCHLD от дочернего процесса о его завершении. Возвращает pid завершившегося процесса, а в аргумент по ссылке кладётся код, с которым потомок завершился. Если на момент вызова дочерних процессов нет, возвращает -1 и 0 соответственно.

**int execlp(const char \*\_\_file, const char \*\_\_arg, ...) <unistd.h>**

Запускает на выполнение файл \_\_file, выполняя поиск в порядке, заданном переменной окружения PATH, если путь до файла не является полным. При запуске передаются аргументы командной строки, указанные в параметрах функции по одному. Обязательным является задание нулевого указателя в конце списка аргументов.

**void exit(int \_\_status) <stdlib.h>**

Выполняет все обработчики, зарегистрированные на завершение процесса, выполняет очистку файловых дескрипторов и завершает программу с кодом завершения \_\_status.

1. **Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#pragma region Global vars

#define FORK\_ERROR 1 // Exit status for failed fork()

#define DATA\_ERROR 2

#pragma endregion

typedef unsigned char byte; // this is missing in standard libs for whatever reason.

struct wait\_t { byte flags; byte code; }; // union for wait result

struct param\_t { // process creation parameters

char \*command; // executable file to be found in PATH

char \*argv0; // argv[0] string, required for exec-() functions

char \*arg1, \*arg2; // first arg, second arg. no more due to complexity

int in, out, err; // descriptors of standard streams

};

// Fill param\_t with process creation data.

void create\_startup\_param(

param\_t \*param, char \*command, char \*arg1, char \*arg2 = NULL,

int out = STDOUT\_FILENO, int err = STDERR\_FILENO, int in = STDIN\_FILENO

) {

param->command = command;

param->argv0 = command;

param->arg1 = arg1;

param->arg2 = arg2;

param->in = in;

param->out = out;

param->err = err;

}

// Overlays current executing process with COMMAND.

int shell\_exec(param\_t \*param) {

dup2(param->out, STDOUT\_FILENO); // identical to fcntl(param->out, F\_DUPFD, 1)

dup2(param->err, STDERR\_FILENO);

dup2(param->in, STDIN\_FILENO);

int ret = execlp(param->command, param->argv0, param->arg1, param->arg2, NULL);

return ((ret == -1) ? -1 : 0); // return -1 on failure, 0 otherwise

}

// Create new thread in new process via fork() and run callback in it.

pid\_t fork\_thread(int(func(param\_t \*param)), void \*param = NULL) {

int childpid;

switch(childpid = fork()) {

case -1:

fprintf(stderr,

"[shell] Could not fork to subcommand '%s'\n", ((param\_t\*)param)->command);

exit(FORK\_ERROR);

break;

case 0:

exit(func((param\_t\*)param)); // run callback and exit

break;

default:

fprintf(stderr,

"[shell] Process %d created for '%s'\n", childpid, ((param\_t\*)param)->command);

return childpid;

break;

}

}

// Wait till ALL running subcommands end

int join\_threads() {

pid\_t wait\_pid;

wait\_t wait\_ret;

int exec\_result = 0;

while((wait\_pid = wait(&wait\_ret)) > 0) {

printf("[shell] Process %d exited with code %d\n", wait\_pid, wait\_ret.code);

exec\_result |= wait\_ret.code; // OR all codes

}

return exec\_result;

}

// Main entry point of application.

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

// We can initialize this struct only once due to fork() behaviour,

// which creates a COPY of process, so memory isn't shared.

// This is why I've set -Wno-write-strings option to gcc.

param\_t \*param = (param\_t\*)malloc(sizeof(param\_t));

int exec\_result;

// Part 1. `cc pr1.c & cc pr2.c` (async run)

create\_startup\_param(param, "cc", "pr1.c"); // cc pr1.c

fork\_thread(shell\_exec, param); // run command

create\_startup\_param(param, "cc", "pr2.c"); // cc pr2.c

fork\_thread(shell\_exec, param); // run command

if((exec\_result = join\_threads()) != 0) // if any of subcommands failed

return exec\_result; // && cat ...

// Part 2. `cat pr1.c pr2.c > prall.c`

int out = open("prall.c", O\_WRONLY | O\_CREAT, 0700); // open/create output file

if(out == -1) { // if attempt to open file has failed

fprintf(stderr, "[shell] Could not open file 'prall.c'\n");

exit(DATA\_ERROR);

}

create\_startup\_param(param, "cat", "pr1.c", "pr2.c", out);//cat pr1.c pr2.c > prall.c

fork\_thread(shell\_exec, param); // run command

return join\_threads(); // exit with return code of last task

}

1. **Результат работы программы**
   1. **pr1.c и pr2 существуют и содержат компилируемый исходный текст**

**[striker@centos7x64 Debug]$** ./lab6\_rm\_self

[shell] Process 22666 created for 'cc'

[shell] Process 22667 created for 'cc'

[shell] Process 22667 exited with code 0

[shell] Process 22666 exited with code 0

[shell] Process 22678 created for 'cat'

[shell] Process 22678 exited with code 0

* 1. **pr2.c содержит ошибки времени компиляции**

**[striker@centos7x64 Debug]$** ./lab6\_rm\_self

[shell] Process 22732 created for 'cc'

[shell] Process 22733 created for 'cc'

**pr2.c:** В функции «**main**»:

**pr2.c:5:2:** ошибка: «**retrurn**» undeclared (first use in this function)

retrurn 0;

^

**pr2.c:5:2:** замечание: each undeclared identifier is reported only once for each function it appears in

**pr2.c:5:10:** ошибка: expected «;» before numeric constant

retrurn 0;

^

[shell] Process 22733 exited with code 1

[shell] Process 22732 exited with code 0

* 1. **В файл prall.c невозможно записывать из-за запрещающих прав доступа**

**[striker@centos7x64 Debug]$** ./lab6\_rm\_self

[shell] Process 22745 created for 'cc'

[shell] Process 22746 created for 'cc'

[shell] Process 22745 exited with code 0

[shell] Process 22746 exited with code 0

[shell] Could not open file 'prall.c'