|  |  |
| --- | --- |
|  | **Отчёт по лабораторной работе** №2 по курсу Операционные системы ­  студента группы М8О-207Б-18 Хренова Геннадия Николаевича , № по списку 28  Адреса www, e-mail, jabber, skype [khrenov.gena@yandex.ru](mailto:khrenov.gena@yandex.ru)  Работа выполнена: “15“ октября 2019 г.  Преподаватель: Миронов Е. С.  Входной контроль знаний с оценкой  Отчёт сдан “ “ 20 г., итоговая оценка  Подпись преподавателя |

1. **Тема**: Процессы, обмен данными между процессорами.
2. **Цель работы**: Приобретение практических навыков в: управление процессами в ОС, Обеспечение обмена данных между процессами посредством каналов.
3. **Задание** (*вариант №* 16): Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы. На вход программе подается команда интерпретатора команд и имя файла. Программа должна перенаправить стандартный ввод команды с этого файла и вывести результат команды в стандартный выходной поток. Использование операций write и printf запрещено.
4. **Оборудование** (*лабораторное*):

ЭВМ Pentium , процессор 2.7 ГГц , имя узла сети с ОП 4096 МБ

НМД 2 ГБ. Терминал адрес . Принтер

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

Процессор intel core i3 , ОП 3072 , НМД 50 ГБ. Монитор

Другие устройства

1. **Программное обеспечение** (*лабораторное*):

Операционная система семейства Unix , наименование версия

Интерпретатор команд bash версия

Система программирования версия

Редактор текстов Emacs версия

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства Unix , наименование Ubuntu версия 18.04LTS

Интерпретатор команд bash версия

Система программирования версия

Редактор текстов Emacs версия

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальное описание с пред- и постусловиями)

Для выполнения работы необходимы следующие команды:

* pid\_t fork() - создание дочернего процесса. Дочерний процесс возвращает 0, при ошибке -1.
* int execlp(const char \**path*, const char \**arg*, ... , (char\*)NULL ) **-** замена образа памяти процесса. Первый параметр, по соглашению, должен указывать на имя, ассоциированное с файлом, который надо запустить. Список параметров *должен* заканчиваться указателем null и, так как это функция с переменным числом аргументов, этот указатель должен приводиться к *(char \*) NULL*. Значение возвращается только в случае ошибки (-1).
* int dup2(int oldfd, int newfd) - переназначение файлового дескриптора. В случае успеха возвращается 0, а в случае ошибки -1.
* Также потребуется создать дескриптор файла. Это можно сделать с помощью системного вызова open(). Первый аргумент - имя файла в файловой системе в обычной форме: полный путь к файлу (если файл не находится в текущем каталоге) или сокращенное имя (если файл в текущем каталоге). Второй аргумент - это режим открытия файла, представляющий собой один или несколько флагов открытия, объединенных оператором побитового ИЛИ.
* pid\_t waitpid(pid\_t pid, int \*status, int options) - Ожидание завершения дочернего процесса

1. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты, либо соображения по тестированию].

Программа получает при вызове 2 аргумента: команду и имя файла. Затем программа создаёт дочерний процесс, там открывает файл и делает перенаправление стандартного ввода из этого файла. После этого необходимо выполнить команду с помощью exec и закрыть файл. Родительский процесс ожидает выполнение дочернего процесса.

Тесты:

1) Подать на вход более двух аргументов

2) команда cat

3) команда head

4) команда ls

5) подать на вход несуществующий файл(файл должен создаться)

*Пункты 1-7 отчёта составляются* ***строго до*** *начала лабораторной работы.*

Допущен к выполнению работы. Подпись преподавателя

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с текстовыми примерами, подписанный преподавателем)

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>

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

if (argc != 3) {

fprintf(stderr, "invalid input\n");

exit(2);

}

pid\_t pid;

int status;

pid = fork();

if (pid < 0) {

fprintf(stderr, "prosses not created\n");

exit(1);

}else if (pid == 0){

int fd = open(argv[2], O\_RDWR | O\_CREAT);

if (fd == -1){

fprintf(stderr, "can`t open file\n");

}

if(dup2(fd, 0) == -1){

fprintf(stderr, "dup eror\n");

}

if (execlp(argv[1], argv[2], NULL) == -1){

fprintf(stderr, "exec eror\n");

}

close(fd);

}else if(pid > 0){

if (waitpid(pid, &status, 0) == -1){

fprintf(stderr, "smth wrong with parent\n");

}

}

return 0;

}

результат работы strace:

execve("./a.out", ["./a.out", "ls", "lol.txt"], 0x7fff156fdbe0 /\* 57 vars \*/) = 0

brk(NULL) = 0x55b6c5f0f000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=90804, ...}) = 0

mmap(NULL, 90804, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fefe179a000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2030544, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fefe1798000

mmap(NULL, 4131552, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fefe1199000

mprotect(0x7fefe1380000, 2097152, PROT\_NONE) = 0

mmap(0x7fefe1580000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7fefe1580000

mmap(0x7fefe1586000, 15072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fefe1586000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7fefe17994c0) = 0

mprotect(0x7fefe1580000, 16384, PROT\_READ) = 0

mprotect(0x55b6c4815000, 4096, PROT\_READ) = 0

mprotect(0x7fefe17b1000, 4096, PROT\_READ) = 0

munmap(0x7fefe179a000, 90804) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fefe1799790) = 2131

wait4(2131, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 2131

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=2131, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0) = ?

+++ exited with 0 +++

Демонстрация работы:

gennadii@lenovo-b560:~/workdir/os$ gcc lab2.c

gennadii@lenovo-b560:~/workdir/os$ ./a.out head lol.txt

first

second

third

4

5

6

7

8

9

10

gennadii@lenovo-b560:~/workdir/os$

**9.Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или  дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. Замечание автора по существу работы
2. Выводы: Для порождения процессов в ОС Linux существует два способа. Один из них позволяет полностью заменить другой процесс, без замены среды выполнения. Другим способом можно создать новый процесс с помощью системного вызова fork(). При вызове fork() порождается новый процесс (процесс-потомок), который почти идентичен порождающему процессу-родителю и наследует многие признаки родителя, такие как сегменты кода, данных, стека программы, таблицу файлов и т.д. Также любой ввод/вывод можно интерпретировать как ввод из некоторого файла и вывод в файл. Работа с файлами производится через их дескрипторы. Для организации ввода/вывода в UNIX используются три файла: stdin (дескриптор 1), stdout (2) и stderr(3).

