Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством технологии “File mapping”.**

Студент: Бутырев Даниил Вячеславович

Группа: М80-206Б-20

Вариант: 5

Преподаватель: Соколов Андрей Алексеевич

Дата:

Оценка:

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

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 memory-mapped files.

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс создаёт дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процессы должны быть представлены разными программами. Родительский процесс передаёт команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

5 вариант) Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**Листинг программы**

module;

#include <signal.h>

#include <sys/types.h>

#include <iostream>

export module signal\_handler;

import explicit\_enum;

import error\_string;

export namespace SignalHandler {

static bool \_all\_good = true;

static pid\_t unit;

void signal\_terminate(int){

\_all\_good = false;

};

void signal\_continue(int){};

void SignalHandler( pid\_t \_unit ) {

unit = \_unit;

signal( SIGUSR1, signal\_continue );

signal( SIGUSR2, signal\_terminate );

}

void terminate() {

int return\_value;

return\_value = kill( unit, SIGUSR2 );

if( return\_value == ReturnValue::bad\_kill ) {

std::cerr << Error::kill\_failed( unit, SIGUSR1 );

exit( ReturnValue::bad\_kill );

}

};

void resume() {

int return\_value;

return\_value = kill( unit, SIGUSR1 );

if( return\_value == ReturnValue::bad\_kill ) {

std::cerr << Error::kill\_failed( unit, SIGUSR1 );

exit( ReturnValue::bad\_kill );

} else {

pause();

}

}

bool all\_good() {

return \_all\_good;

}

};

module;

#include <string>

export module error\_string;

export namespace Error {

std::string forking\_process( const int pid ) {

std::string return\_str = "error: forking process: ";

return return\_str + std::to\_string( pid );

}

std::string executing\_program( const std::string& pathname ) {

std::string return\_str = "error: executing program: ";

return return\_str + pathname;

}

std::string mmap\_failed( const int mapping\_length ) {

std::string return\_str = "error: mmap memory with length: ";

return return\_str + std::to\_string( mapping\_length );

}

std::string kill\_failed( pid\_t pid, int signal ) {

std::string return\_str = "error: kill failed : pid = ";

return return\_str + std::to\_string( static\_cast< int >( pid ) ) +

" and signal = " + std::to\_string( signal );

}

std::string opening\_file( const std::string& file ) {

std::string return\_str = "error: open failed, file: ";

return return\_str + file;

}

std::string writing\_to\_file( const int fd, const std::string& file ) {

std::string return\_str = "error: writing to file with fd: ";

return return\_str + std::to\_string( fd ) + " and sentence: " + file;

}

};

export module explicit\_enum;

export namespace StandardValue {

enum {

bits\_in\_byte = 8,

array\_begin = 0,

};

};

export namespace ReturnValue {

enum standard {

nice = 0,

};

enum exec {

bad\_exec = -1,

};

enum mmap {

mmap\_failed,

};

enum fork {

bad\_fork = -1,

fork\_child\_program = 0,

};

enum kill {

bad\_kill = -1,

};

enum open {

open\_fail = -1,

};

enum write {

write\_fail = -1,

};

enum error {

error\_executing\_program = 1,

error\_forking\_process,

error\_opening\_file,

error\_not\_enough\_args,

error\_writing\_to\_file,

};

};

export namespace MmapValue {

enum flags { //exactly

anon\_fd = -1,

anon\_offset = 0,

no\_offset = 0,

};

};

export namespace ExecValue {

enum args {

here\_not\_zero = 1,

here\_zero = 2,

not\_zero = 1,

};

extern const auto last\_arg = nullptr;

enum pointer {

program\_name = 0,

pointer\_arg = 1,

mask\_pointer\_arg = 2,

};

};

// enum {

// read\_end=0,

// write\_end=1,

// };

// enum {

// stdin = 0,

// stdout = 1,

// stderr = 2,

// };

// error\_creating\_pipe,

// error\_child\_args,

// error\_creating\_process,

// error\_reading\_input,

// error\_writing\_pipe,

// error\_reading\_pipe,

// error\_writing\_err,

// };

module;

#include <memory>

#include <type\_traits>

#include <cstdint>

#include <string>

#include <iostream>

export module exec\_arg;

import explicit\_enum;

export namespace ExecArg {

template < typename T >

void make\_execve\_arg( T pointer, std::string& pathname,

char\*\*& exec\_arg ) {

std::allocator< char > alloc\_ch;

std::cout << "making pointer: |" << pointer << "|\n";

const int exec\_arg\_count = 4; //name, pointer, mask, null

char\* pointer\_arg = alloc\_ch.allocate( 2 \* ( sizeof( T ) + sizeof( '\0' ) ) );

const int mask\_offset = sizeof( T ) + sizeof( '\0' );

for( decltype( sizeof( T ) ) i = StandardValue::array\_begin;

i < sizeof( T ); ++i) {

// std::cout << i << " byte: " << \*( reinterpret\_cast< unsigned char\* >( &pointer ) + i );

pointer\_arg[ i ] = \*( reinterpret\_cast< unsigned char\* >( &pointer ) + i );

pointer\_arg[ i + mask\_offset ] = ExecValue::here\_not\_zero;

if( pointer\_arg[ i ] == '\0' ) {

pointer\_arg[ i ] = ExecValue::not\_zero;

pointer\_arg[ i + mask\_offset ] = ExecValue::here\_zero;

}

}

pointer\_arg[ sizeof( T ) ] = '\0';

pointer\_arg[ mask\_offset + sizeof( T ) ] = '\0';

std::allocator< char\* > alloc\_chptr;

exec\_arg = alloc\_chptr.allocate( exec\_arg\_count );

exec\_arg[ ExecValue::program\_name ] = pathname.data();

exec\_arg[ ExecValue::pointer\_arg ] = pointer\_arg;

// std::cout << "have pointer intptr\_t: " << \*reinterpret\_cast<std::intptr\_t\*>(&pointer) << '\n';

// std::cout << "throw pointer intptr\_t: " << \*reinterpret\_cast<std::intptr\_t\*>(pointer\_arg) << '\n';

// std::cout << "pointer throw: ";

// for(int i = 0; i < sizeof(void\*); ++i) {

// std::cout << static\_cast<int>(static\_cast<unsigned char>(pointer\_arg[i])) << ' ';

// }

// std::cout <<'\n';

// std::cout << "mask throw: ";

// for(int i = 0; i < sizeof(void\*); ++i) {

// std::cout << static\_cast<int>(pointer\_arg[i + mask\_offset]) << ' ';

// }

// std::cout <<'\n';

exec\_arg[ ExecValue::mask\_pointer\_arg ] = pointer\_arg + mask\_offset;

exec\_arg[ exec\_arg\_count - 1 ] = ExecValue::last\_arg; //last should be null

}

template <typename T>

void get\_execve\_arg( T& pointer, char\*\* exec\_arg ) {

// std::cout << "catch pointer intptr\_t: " << \*reinterpret\_cast<std::intptr\_t\*>(exec\_arg[ExecValue::pointer\_arg]) << '\n';

// std::cout << "pointer catch: ";

// for(int i = 0; i < sizeof(void\*); ++i) {

// std::cout << static\_cast<int>(static\_cast<unsigned char>(exec\_arg[ExecValue::pointer\_arg][i])) << ' ';

// }

// std::cout <<'\n';

// std::cout << "mask catch: ";

// for(int i = 0; i < sizeof(void\*); ++i) {

// std::cout << static\_cast<int>(static\_cast<unsigned char>(exec\_arg[ExecValue::mask\_pointer\_arg][i])) << ' ';

// }

// std::cout <<'\n';

//std::intptr\_t pointer\_raw = 0;

//1 var

// for( decltype( sizeof( void\* ) ) i = StandardValue::array\_begin;

// i < sizeof( void\* ); ++i ) {

// pointer\_raw <<= StandardValue::bits\_in\_byte;

// if( exec\_arg[ ExecValue::mask\_pointer\_arg ][ i ] != ExecValue::here\_zero ) {

// pointer\_raw += static\_cast< unsigned char >( exec\_arg[ ExecValue::pointer\_arg ][ i ] );

// }

// std::cout << \*reinterpret\_cast< T\*\* >( &pointer\_raw ) << '\n';

// }

// 2 var

// pointer\_raw = \*reinterpret\_cast<std::intptr\_t\*>(exec\_arg[ExecValue::pointer\_arg]);

// for( decltype( sizeof( void\* ) ) i = StandardValue::array\_begin;

// i < sizeof( void\* ); ++i ) {

// if( exec\_arg[ ExecValue::mask\_pointer\_arg ][ i ] == ExecValue::here\_zero ) {

// \*(reinterpret\_cast<unsigned char\*>( &pointer\_raw ) + i ) = 0;

// }

// }

// pointer = \*reinterpret\_cast< T\*\* >( &pointer\_raw );

//

char pointer\_raw[ sizeof( T ) ];

for( decltype( sizeof( T ) ) i = 0; i < sizeof( T ); ++i ) {

pointer\_raw[ i ] = exec\_arg[ ExecValue::pointer\_arg ][ i ];

if( exec\_arg[ ExecValue::mask\_pointer\_arg ][ i ] == ExecValue::here\_zero ) {

pointer\_raw[ i ] = 0;

}

}

pointer = \*reinterpret\_cast< T\* >( pointer\_raw );

std::cout << "have got pointer: |" << pointer << "|\n";

// std::cout << "pointer intptr\_t: " << pointer\_raw << '\n';

}

};

module;

#include <sys/mman.h>

#include <iostream>

#include <fstream>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string>

export module mmap;

import error\_string;

import explicit\_enum;

export namespace Mmap {

enum {

dont\_have\_fd = -1,

};

template<typename T>

void get\_mapped\_area( int& fd, const int mapping\_length, T\*& mapped\_area ) {

const char sentence[] = "fucking fuck stupid feature";

const char directory\_for\_tmp\_file[] = ".";

if( fd == Mmap::dont\_have\_fd ) {

fd = open( directory\_for\_tmp\_file, O\_TMPFILE | O\_RDWR,

O\_TMPFILE | S\_IRWXU );

if( fd == ReturnValue::open\_fail ) {

std::string filename = "temporary file in directory";

std::cerr << Error::opening\_file( filename +

directory\_for\_tmp\_file );

exit( ReturnValue::error\_opening\_file );

}

if( write( fd, sentence, sizeof( sentence ) ) ==

ReturnValue::write\_fail ) { // Why I should do this WTF

std::cerr << // otherwise I get Ошибка шины (стек памяти сброшен на диск)

Error::writing\_to\_file( fd, std::string( sentence ) );

exit( ReturnValue::error\_writing\_to\_file );

}

}

// if( fd == 0 ) {

// fd = open( ".some\_unique\_file", O\_CREAT | O\_TRUNC | O\_RDWR, O\_CREAT | S\_IRWXU );

// write( fd, sentence, sizeof( sentence ) ); // Why I should do this WTF

// // otherwise I get Ошибка шины (стек памяти сброшен на диск)

// if( fd == -1 ) {

// std::cerr << "can't open file";

// exit(1);

// }

// }

mapped\_area = static\_cast< T\* >( mmap( NULL, mapping\_length,

PROT\_READ | PROT\_WRITE | PROT\_EXEC,

MAP\_SHARED,

fd, MmapValue::no\_offset ) );

if ( mapped\_area == MAP\_FAILED ) {

std::cerr << Error::mmap\_failed( mapping\_length );

exit( ReturnValue::mmap\_failed );

}

}

};

#include <sys/types.h>

#include <unistd.h>

#include <string>

#include <iostream>

#include <sys/mman.h>

import explicit\_enum;

import error\_string;

import read\_input;

import mmap;

import exec\_arg;

import signal\_handler;

int main() {

std::string pathname;

ReadInput::read\_string( pathname );

int\* mapped\_area = nullptr;

const int mapping\_length = 100;

int fd = Mmap::dont\_have\_fd;

Mmap::get\_mapped\_area< int >( fd, mapping\_length, mapped\_area );

std::cout << "parent: mapped\_area = " << mapped\_area << std::endl;

char\*\* exec\_arg;

ExecArg::make\_execve\_arg< int >( fd, pathname, exec\_arg );

// for( int i = 0; i <= pathname.size(); ++i ) {

// std::cout << static\_cast< int >( pathname[ i ] ) << '\n';

// }

// std::cout << '\n';

// for( int i = 0; exec\_arg[2][i] != '\0'; ++i ) {

// std::cout << static\_cast< int >( exec\_arg[2][ i ] ) << '\n';

// }

pid\_t child\_pid = fork();

if( child\_pid == ReturnValue::bad\_fork ) {

std::cerr << Error::forking\_process( getpid() );

return ReturnValue::error\_forking\_process;

} else if( child\_pid == ReturnValue::fork\_child\_program ) {

int exec\_return\_value = execv( pathname.c\_str(), exec\_arg );

if( exec\_return\_value == ReturnValue::bad\_exec ) {

std::cerr << Error::executing\_program( pathname );

return ReturnValue::error\_executing\_program;

}

}

SignalHandler::SignalHandler( child\_pid );

bool all\_good = true;

int number = 0;

//SignalHandler::resume();

while( all\_good ) {

std::cin >> number;

if( !std::cin.good() ) {

all\_good = false;

SignalHandler::terminate();

} else {

std::cout << "read number: " << number;

mapped\_area[ 0 ] = number;

std::cout << "; mapped\_area = " << mapped\_area[ 0 ] << std::endl;

SignalHandler::resume();

all\_good = SignalHandler::all\_good();

}

}

munmap(mapped\_area, mapping\_length);

close( fd );

return ReturnValue::nice;

}

module;

#include <string>

#include <iostream>

export module read\_input;

export namespace ReadInput {

void read\_string( std::string& str ) {

std::cin >> str;

}

};

module;

#include <set>

#include <cmath>

export module prime\_number;

void put\_all\_primes\_until( std::set<int>& primes, int num );

export bool is\_prime( const int& number ) {

static std::set< int > primes{ 2, 3, 5, 7, 11, 13 };

if( number < 2 ) return false;

if( primes.count( number ) ) return true;

int sqrt\_num = std::trunc( std::sqrt( number ) );

if( sqrt\_num > \*primes.rbegin() )

put\_all\_primes\_until( primes, sqrt\_num );

for( auto another\_prime : primes )

if( number % another\_prime == 0 )

return false;

primes.insert( number );

return true;

}

void put\_all\_primes\_until( std::set<int>& primes, int num ) {

int max\_exist = \*( primes.rbegin() );

if( max\_exist > num ) return;

for( int i = ( max\_exist - max\_exist % 6 ) + 6; i <= num; i += 6 ) {

is\_prime( i - 1 );

is\_prime( i + 1 ); //it pushes them for u

}

}

#include <fstream>

#include <sys/types.h>

#include <unistd.h>

#include <string>

#include <iostream>

import mmap;

import exec\_arg;

import prime\_number;

import explicit\_enum;

import signal\_handler;

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

int fd;

const int mapping\_length = 100;

int\* mapped\_area = nullptr;

if( argc != 3 ) //name, ptr, ptr\_mask

return ReturnValue::error\_not\_enough\_args;

ExecArg::get\_execve\_arg< int >( fd, argv );

std::fstream file( "numbers.ans", std::ios::trunc | std::ios::out );

int number = 0;

bool all\_good = true;

std::cout << "accessing signal handler:..." << std::endl;

SignalHandler::SignalHandler( getppid() );

Mmap::get\_mapped\_area(fd, mapping\_length, mapped\_area);

std::cout << "child: mapped\_area = " << mapped\_area << '\n';

SignalHandler::resume();

all\_good = SignalHandler::all\_good();

while( all\_good ) {

std::cout << "accessing mapped\_area:..." << std::endl;

number = mapped\_area[ 0 ];

std::cout << "success: " << number << std::endl;

bool negative = number < 0;

bool prime = false;

if( !negative ) {

prime = is\_prime(number);

}

bool num\_is\_correct = !( negative || prime );

if( num\_is\_correct ) {

std::cout << "correct" << std::endl;

file << std::to\_string(number) << std::endl;

SignalHandler::resume();

} else {

std::cout << "incorrect" << std::endl;

SignalHandler::terminate();

all\_good = false;

}

all\_good = SignalHandler::all\_good() && all\_good;

}

file.close();

return ReturnValue::nice;

}

**Примеры работы**

steep@gg:~/educat/instit/os/4lab/src$ ./parent

child

parent: mapped\_area = 0x7f1ae2a4e000

making pointer: |3|

have got pointer: |3|

accessing signal handler:...

child: mapped\_area = 0x7f49cc036000

100

read number: 100; mapped\_area = 100

accessing mapped\_area:...

success: 100

correct

200

read number: 200; mapped\_area = 200

accessing mapped\_area:...

success: 200

correct

300

read number: 300; mapped\_area = 300

accessing mapped\_area:...

success: 300

correct

3

read number: 3; mapped\_area = 3

accessing mapped\_area:...

success: 3

incorrect

steep@gg:~/educat/instit/os/4lab/src$ cat numbers.ans

100

200

300

**Вывод**

Для взаимодействия между процессами можно использовать технологию file mapping, которая позволяет работать с разделяемыми между процессами данными как с массивом.