Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Ергизов А. Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата:20.11.25

Москва, 2025

Постановка задачи

Вариант 5.

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 это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются

Общий метод и алгоритм решения

Использованные системные вызовы работы с разделяемой памятью:

- int shm_open(const char *name, int oflag, mode_t mode);
  Создает или открывает объект разделяемой памяти, возвращает файловый дескриптор.
- int ftruncate(int fd, off_t length);
  Устанавливает размер файла/SHM (например, увеличивает до SHM_SIZE).
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
  Отображает файл/SHM в адресное пространство процесса.
- int munmap(void *addr, size_t length);
  Убирает отображение памяти из адресного пространства.
- int shm_unlink(const char *name);
  Удаляет объект разделяемой памяти из системы.
- int close(int fd);
  Закрывает файловый дескриптор SHM (или любого файла).

Использованные системные вызовы работы с семафорами:

- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);

  Создает или открывает именованный семафор.

- int sem_wait(sem_t *sem);

  Уменьшает счетчик семафора или блокируется, если он равен 0.

- int sem_post(sem_t *sem);

  Увеличивает счетчик семафора, разблокируя ожидающие процессы.

- int sem_close(sem_t *sem);

  Закрывает дескриптор семафора для текущего процесса.

- int sem_unlink(const char *name);

  Удаляет именованный семафор из системы.

Использованные системные вызовы работы с процессами:

- pid_t fork(void);

  Создает новый процесс, копию текущего.

- int execv(const char *path, char *const argv[]);

  Загружает новый исполняемый файл в процесс, заменяя его.

- pid_t waitpid(pid_t pid, int *status, int options);

  Ждет завершения конкретного дочернего процесса или проверяет его статус.

- void _exit(int status);

  Немедленно завершает процесс с заданным статусом, минуя atexit и буферы stdio.

Использованные системные вызовы работы с вводом- выводом:

- ssize_t write(int fd, const void *buf, size_t count);

  Пишет данные в файл/терминал/SHM (любой дескриптор).

- ssize_t read(int fd, void *buf, size_t count);

  Считывает данные из файла/терминала/SHM (любой дескриптор).

- int open(const char *pathname, int flags, mode_t mode);

  Открывает или создает файл для записи/чтения.

В ходе лабораторной работы я создавал объекты разделяемой памяти, отображал их в адресное пространство процессов. Создавал процессы, что бы они могли общаться с помощью той самой разделяемой памяти для обработки входных данных. Для избежания гонок использовал семафор.

Код программы

Parent.c

```
#include <fcntl.h>

#include <stdint.h>
#include <stdbool.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
```

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <semaphore.h>

#include <stdio.h>

#define SHM_SIZE 4096

char SHM_NAME[1024];
char SEM_EMPTY[1024];
char SEM_FULL[1024];

bool is_prime(int n)
{
if (n < 2)
return false;
if (n == 2)
return true;
if (n % 2 == 0)
return false;
for (int i = 3; i * i <= n; i += 2)
if (n % i == 0)
return false;
return true;
}

int main()
{
char unique[64];
snprintf(unique, sizeof(unique), "%d_%ld", getpid(), (long)time(NULL));
snprintf(SHM_NAME, sizeof(SHM_NAME), "/shm_%s", unique);
snprintf(SEM_EMPTY, sizeof(SEM_EMPTY), "/sem_empty_%s", unique);
snprintf(SEM_FULL, sizeof(SEM_FULL), "/sem_full_%s", unique);

int shm_fd = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0600);
if (shm_fd == -1)
{
const char msg[] = "error: failed to create SHM\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}

if (ftruncate(shm_fd, SHM_SIZE) == -1)
{
const char msg[] = "error: failed to resize SHM\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}
```

```
int *shared_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
if (shared_data == MAP_FAILED)
{
const char msg[] = "error: failed to map SHM\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}

sem_t *sem_empty = sem_open(SEM_EMPTY, O_CREAT | O_EXCL, 0600, 1);
sem_t *sem_full = sem_open(SEM_FULL, O_CREAT | O_EXCL, 0600, 0);
if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED)
{
const char msg[] = "error: failed to create semaphore\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}

pid_t child = fork();
if (child == 0)
{
char *args[] = {"Child", SHM_NAME, SEM_EMPTY, SEM_FULL, NULL};
execv("./Child", args);
const char msg[] = "error: failed to exec child\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}
else if (child < 0)
{
const char msg[] = "error: failed to fork\n";
write(STDERR_FILENO, msg, sizeof(msg));
_exit(EXIT_FAILURE);
}

bool running = true;
char buffer[128];
while (running)
{
const char prompt[] = "Enter number: ";
write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
if (bytes <= 0)
break;
buffer[bytes - 1] = '\0';

int number = atoi(buffer);

sem_wait(sem_empty);
```

```
shared_data[0] = number;
sem_post(sem_full);

if (number < 0 || is_prime(number))
running = false;
}

sem_wait(sem_empty);
shared_data[0] = -2;
sem_post(sem_full);

waitpid(child, NULL, 0);

sem_close(sem_empty);
sem_close(sem_full);
sem_unlink(SEM_EMPTY);
sem_unlink(SEM_FULL);

munmap(shared_data, SHM_SIZE);
shm_unlink(SHM_NAME);
close(shm_fd);

return 0;
}
```

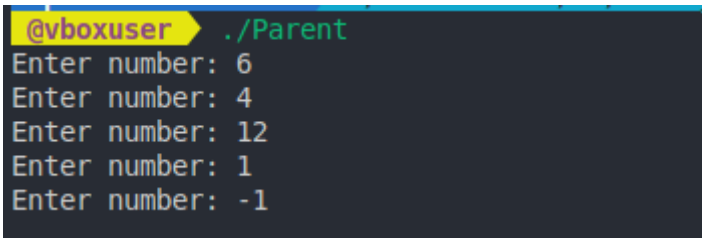
Child.c

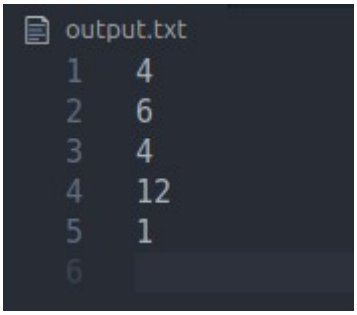
```
#include <fcntl.h>

#include <stdint.h>
#include <stdbool.h>

#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <unistd.h>
#include <sys/mman.h>
#include <semaphore.h>

#include <stdio.h>

#define SHM_SIZE 4096

bool is_prime(int n)
{
if (n < 2)
return false;
```

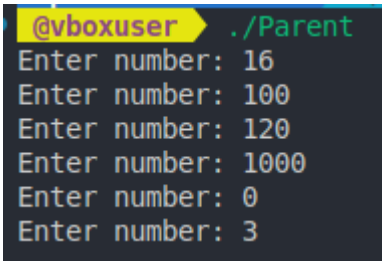
```
if (n == 2)
return true;
if (n % 2 == 0)
return false;
for (int i = 3; i * i <= n; i += 2)
if (n % i == 0)
return false;
return true;
}

int main(int argc, char *argv[])
{
if (argc < 4)
_exit(EXIT_FAILURE);

char *shm_name = argv[1];
char *sem_empty_name = argv[2];
char *sem_full_name = argv[3];

int shm_fd = shm_open(shm_name, O_RDWR, 0600);
if (shm_fd == -1)
_exit(EXIT_FAILURE);

int *shared_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
if (shared_data == MAP_FAILED)
_exit(EXIT_FAILURE);

sem_t *sem_empty = sem_open(sem_empty_name, 0);
sem_t *sem_full = sem_open(sem_full_name, 0);
if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED)
_exit(EXIT_FAILURE);

int fd_output = open("output.txt", O_CREAT | O_WRONLY | O_APPEND, 0644);
if (fd_output == -1)
_exit(EXIT_FAILURE);

bool running = true;
while (running)
{
sem_wait(sem_full);

int number = shared_data[0];
if (number == -2 || number < 0 || is_prime(number))
{
sem_post(sem_empty);
break;
}

char buf[32];
```

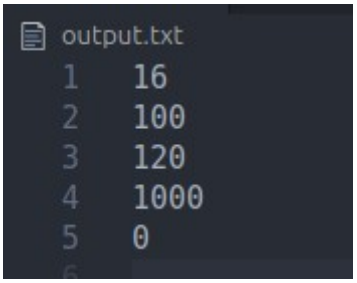
```
int n = snprintf(buf, sizeof(buf), "%d\n", number);
write(fd_output, buf, n);

sem_post(sem_empty);
}

close(fd_output);
sem_close(sem_empty);
sem_close(sem_full);
munmap(shared_data, SHM_SIZE);
close(shm_fd);

return 0;
}
```

Протокол работы программы









Вывод

В ходе лабораторной работы я приобрел практические навыки при работе с разделяемой памятью, с ее отображением на адресное пространство процесса, а также потренировался в использовании семафоров.