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-210БВ-24

Студент: Телепнева А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.11.25

Москва, 2025

Постановка задачи

Вариант 19.

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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- pid_t wait(int *status) — ожидает завершения дочернего процесса.
- int shm_open(const char *name, int oflag, mode_t mode) — создаёт или открывает именованный объект разделяемой памяти.
- int ftruncate(int fd, off_t length) — задаёт размер объекта разделяемой памяти.
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) — отображает объект разделяемой памяти в адресное пространство процесса.
- int munmap(void *addr, size_t length) — удаляет отображение разделяемой памяти.
- int shm_unlink(const char *name) — удаляет именованный объект разделяемой памяти
- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value) — создаёт или открывает именованный семафор.
- int sem_wait(sem_t *sem) — уменьшает значение семафора (ожидание).
- int sem_post(sem_t *sem) — увеличивает значение семафора (сигнал).
- int sem_close(sem_t *sem) — закрывает семафор.
- int sem_unlink(const char *name) — удаляет именованный семафор.
- int open(const char *pathname, int flags, mode_t mode) — открывает файл для записи результатов.
- ssize_t write(int fd, const void *buf, size_t count) — записывает данные в файл.
- int close(int fd) — закрывает файловый дескриптор.

Описание работы программы:
Алгоритм работы родительского процесса (server.c):

1. Формируется уникальное имя для объекта разделяемой памяти и семафоров.

2. Создаётся объект разделяемой памяти с помощью shm_open, задаётся его размер через ftruncate.
3. Разделяемая память отображается в адресное пространство процесса с помощью mmap.
4. Создаются два именованных семафора:
    - семафор записи, инициализированный значением 1;
    - семафор чтения, инициализированный значением 0.
5. С помощью fork создаётся дочерний процесс.
6. Родительский процесс считывает строки, введённые пользователем.
7. Перед записью данных в разделяемую память родитель выполняет sem_wait на семафоре записи.
8. Строка записывается в разделяемую память.
9. Родительский процесс сигнализирует дочернему процессу о готовности данных с помощью sem_post семафора чтения.
10. После получения сигнала завершения ввода родитель ожидает завершения дочернего процесса с помощью wait.
11. Освобождаются все ресурсы: разделяемая память и семафоры удаляются.

Алгоритм работы дочернего процесса (client.c):

1. Дочерний процесс получает доступ к уже созданной разделяемой памяти и семафорам.
2. Открывается файл для записи результатов обработки.
3. Дочерний процесс ожидает появления данных в разделяемой памяти, вызывая sem_wait на семафоре чтения.
4. После получения строки данные считываются из разделяемой памяти.
5. Из строки удаляются все гласные буквы.
6. Результат записывается в файл.
7. Дочерний процесс сигнализирует родительскому процессу о завершении обработки с помощью sem_post семафора записи.
8. При получении признака завершения ввода дочерний процесс закрывает файл и корректно завершает работу.

Код программы

server.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#include <time.h>


#define BUF_SIZE 4096


typedef struct {

    char buffer[BUF_SIZE];

    int terminate;

} shm_data;


void create_ipc(const char *shm_name, const char *sem_empty_name, const char
*sem_full_name, shm_data **data, sem_t **sem_empty, sem_t **sem_full) {

    int shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0600);

    ftruncate(shm_fd, sizeof(shm_data));


    *data = mmap(NULL, sizeof(shm_data), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);


    (*data)->terminate = 0;


    *sem_empty = sem_open(sem_empty_name, O_CREAT, 0600, 1);

    *sem_full  = sem_open(sem_full_name,  O_CREAT, 0600, 0);

}
```

```
int main() {
    srand(time(NULL));
    pid_t pid = getpid();

    char shm1[64], shm2[64];
    char sem1_empty[64], sem1_full[64];
    char sem2_empty[64], sem2_full[64];

    snprintf(shm1, sizeof(shm1), "/shm_%d_1", pid);
    snprintf(shm2, sizeof(shm2), "/shm_%d_2", pid);

    snprintf(sem1_empty, sizeof(sem1_empty), "/sem_%d_1_e", pid);
    snprintf(sem1_full,  sizeof(sem1_full),  "/sem_%d_1_f", pid);

    snprintf(sem2_empty, sizeof(sem2_empty), "/sem_%d_2_e", pid);
    snprintf(sem2_full,  sizeof(sem2_full),  "/sem_%d_2_f", pid);

    shm_data *d1, *d2;
    sem_t *e1, *f1, *e2, *f2;

    create_ipc(shm1, sem1_empty, sem1_full, &d1, &e1, &f1);
    create_ipc(shm2, sem2_empty, sem2_full, &d2, &e2, &f2);

    if (fork() == 0) {
        execl("./client", "client", shm1, sem1_empty, sem1_full, "out1.txt", NULL);
        exit(1);
    }

    if (fork() == 0) {
```

```
    execl("./client", "client", shm2, sem2_empty, sem2_full, "out2.txt", NULL);
    exit(1);
}

char buf[BUF_SIZE];

while (fgets(buf, sizeof(buf), stdin)) {
    int r = rand() % 100;

    shm_data *d   = (r < 80) ? d1 : d2;
    sem_t   *emp = (r < 80) ? e1 : e2;
    sem_t   *ful = (r < 80) ? f1 : f2;

    sem_wait(emp);
    strcpy(d->buffer, buf);
    sem_post(ful);
}

sem_wait(e1);
d1->terminate = 1;
sem_post(f1);

sem_wait(e2);
d2->terminate = 1;
sem_post(f2);

sleep(1);

shm_unlink(shm1);
shm_unlink(shm2);
```

```
    sem_unlink(sem1_empty);
    sem_unlink(sem1_full);
    sem_unlink(sem2_empty);
    sem_unlink(sem2_full);

    return 0;
}
```

client.c

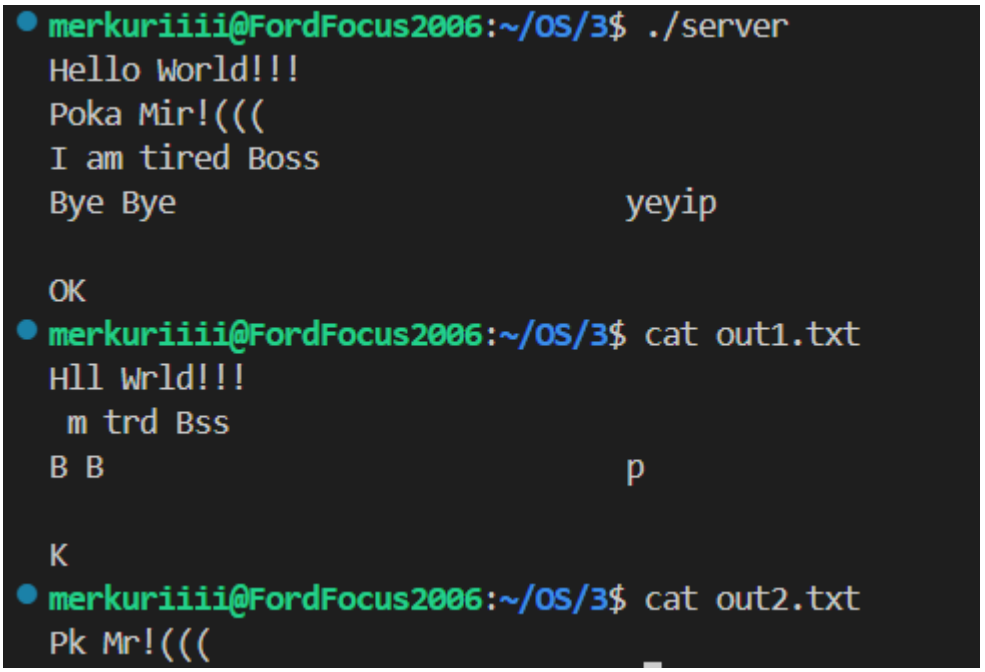
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <string.h>

#define BUF_SIZE 4096

typedef struct {
    char buffer[BUF_SIZE];
    int terminate;
} shm_data;

int is_vowel(char c) {
    return strchr("aeiouyAEIOUY", c) != NULL;
}

int main(int argc, char **argv) {
    char *shm_name = argv[1];
```

```
    char *sem_empty_name = argv[2];
    char *sem_full_name  = argv[3];
    char *filename = argv[4];

    int shm_fd = shm_open(shm_name, O_RDWR, 0600);
    shm_data *data = mmap(NULL, sizeof(shm_data), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

    sem_t *sem_empty = sem_open(sem_empty_name, 0);
    sem_t *sem_full  = sem_open(sem_full_name,  0);

    int file = open(filename, O_CREAT | O_WRONLY | O_TRUNC, 0644);

    while (1) {
      sem_wait(sem_full);

      if (data->terminate)
        break;

      char out[BUF_SIZE];
      int j = 0;

      for (int i = 0; data->buffer[i]; i++)
        if (!is_vowel(data->buffer[i])) {
          out[j++] = data->buffer[i];
        }

      write(file, out, j);
      sem_post(sem_empty);
    }
```

```
    close(file);
    munmap(data, sizeof(shm_data));
    sem_close(sem_empty);
    sem_close(sem_full);

    return 0;
}
```

Протокол работы программы

Тестирование:



Вывод

В ходе выполнения лабораторной работы была реализована система межпроцессного взаимодействия с использованием именованной разделяемой памяти и семафоров в ОС Linux.
Была обеспечена корректная синхронизация процессов без применения активного ожидания.
По сравнению с первой лабораторной, в данной работе достигнута более эффективная и масштабируемая организация обмена данными между процессами.

Программа корректно обрабатывает вводимые данные, освобождает все используемые системные ресурсы и завершается без ошибок.