Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 6

на тему «Элементы сетевого программирования»

Выполнил             А. К. Хрищанович

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 6](#_Toc157722975)

[Выводы 7](#_Toc157722976)

[Список использованных источников 8](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 9](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является построение системы обмена файлами клиентами через сеть с возможностью выбора и отправки файлов.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сеть – это типичная сложная система, которая состоит из подсистем, которые в свою очередь также являются сложными системами.

Существует два основных уровня компонентов сети:

- базовая сеть передачи данных;

- оконечные узлы.

Существует два основных способа передачи данных:

1. С установлением соединения: обеспечение целостности и упорядоченности потока передаваемых данных. Данные доставляются строго в том порядке, в котором были отправлены. Прерывание потока своевременно распознается. Это достигается за счет нумерации порций данных и организации встречного потока подтверждений о получении (квитанций). Образуется виртуальный канал передачи данных, который близок по свойствам к файлу или потоку ввода-вывода.

2. Без установления соединения: Данные передаются в виде датаграмм. Каждая датаграмма доставляется к получателю по произвольному маршруту и независимо от других датаграмм. Подтверждение о получении не предусматривается, поэтому не гарантируется ни порядок следования датаграмм, ни единственность доставленного экземпляра, ни сам факт доставки. Контролируются только искажения каждой отдельной датаграммы.

Открытая система – это система, построенная на основе открытых спецификаций, то есть таких, которые доступны для общего пользования и изменения. В контексте сетей, открытость есть доступность средств взаимодействия компонентов сети для общего использования и совместимость с другими системами.

Модель взаимодействия открытых систем OSI – это стандартизированная модель, разработанная Международной организаций по стандартизации ISO в 1983 году для описания взаимодействия различных устройств в сети.

Модель OSI разделяет коммуникационные функции в сети на уровни, начиная от физического соединения и заканчивая прикладными приложениями. Каждый уровень выполняет определенные функции, а взаимодействие между уровнями осуществляется через стандартизированные протоколы.

Сети, как правило, организованы иерархически, где различные уровни предоставляют разные уровни абстракции и функциональности. Это помогает в масштабировании и обслуживании сети.

Унификация в контексте сетей означает использование стандартных протоколов и спецификаций для обеспечения совместимости и взаимодействия между различными системами и компонентами сети.

Протокол – набор правил и процедур взаимодействия между одноименными уровнями различных систем, которые обеспечивают корректную связь участников взаимодействия в сети.

Интерфейс – набор правил и средств их реализации для взаимодействия между соседними уровнями одной системы, которые обеспечивают возможность модульного построения системы.

Стек протоколов в сети – набор протоколов, обслуживающих различные уровни взаимодействия. Протоколы в стеке проектируются с расчетом на совместную согласованную работу, но остаются достаточно независимыми для возможности замены на альтернативные с сохранением интерфейсов.

Сокет – программный объект, который обычно предоставляется операционной системой для создания сетевых соединений. Сокет скрывает детали реализации доступа к системе, позволяя приложениям взаимодействовать через сеть.

Реализация сокетов на операционной системе Windows и Unix отличаются. На Windows сокеты реализуются при помощи подсистемы WinSock и соответствующих библиотек. На Unix же сокеты обычно реализованы в ядре и для их использования необходимы иные библиотеки.

Для выполнения данной лабораторной работы были использованы следующие сведения и концепции:

1 Разделяемая память: для работы с разделяемой памятью была использована структура данных, к которой обращались процессы,   
а также функции ftok для создания ключа, shmget для создания нового сегмента разделяемой памяти, shmat для присоединения сегмента разделяемой памяти, shmdt для отсоединения сегмента разделяемой памяти, shmctl   
для удаления сегмента разделяемой памяти.

2 Семафоры: для контроля доступа к разделяемой памяти каждого процесса были использованы семафоры, а также функции malloc   
для выделения определенного размера памяти под семафор, sem\_init   
для инициализации семафора, sem\_wait для уменьшения значения семафора на единицу, sem\_post для увеличения значения семафора на единицу, sem\_destroy для уничтожения семафора, free для освобождения выделенной памяти под семафор.

3 Управление процессами: при помощи функции fork создавались дочерние процессы, в которых происходила инициализация значениями блока данных. После при помощи функции wait родительский процесс ждет завершения дочерних процессов. Подсчет суммы значений блока данных осуществляется уже в основном родительском процессе.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе лабораторной работы была реализована программа, в которой реализуется создание нескольких дочерних процессов, которые заполняют случайными значениями определенный блок данных, после чего родительский процесс подсчитывает сумму этих значений. Результат работы программы представлен на рисунке 3.1.

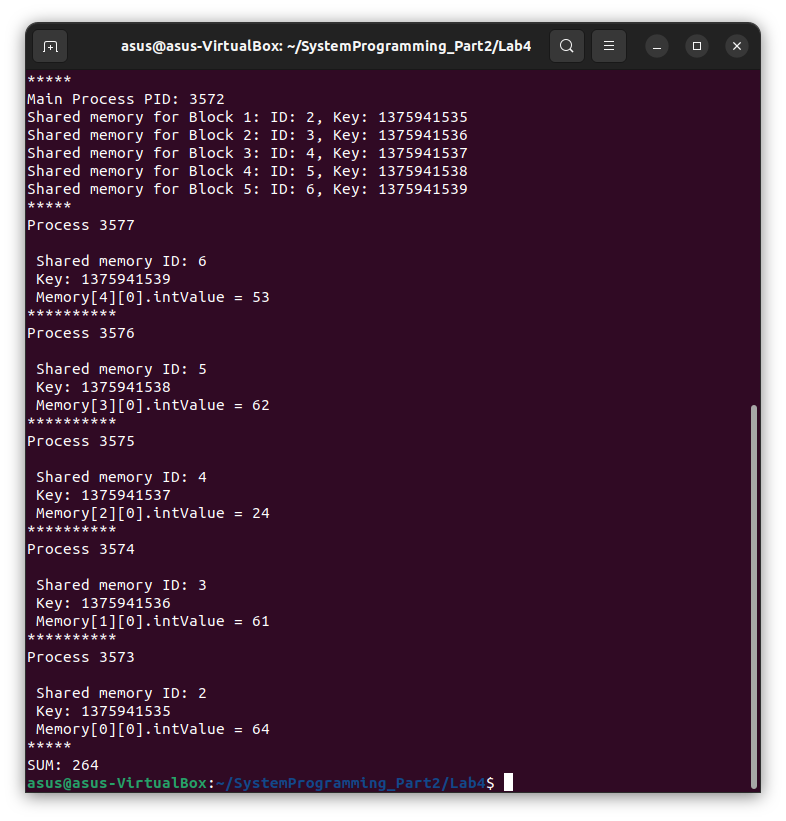


Рисунок 3.1 – Результат работы программы

Таким образом, в ходе лабораторной работы была реализована программа, реализующая заполнение блока данных случайными значениями и подсчет суммы этих значений.

# **ВЫВОДЫ**

В ходе лабораторной работы были изучены основные особенности подсистемы управления процессами и средства взаимодействия процессов   
в Unix. Кроме этого, была реализована программа на языке программирования С, которая параллельно обрабатывает блок данных различными процессами, которые заполняют данный блок данных случайными числами, после чего родительский процесс подсчитывает сумму этих значений.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Процессы и потоки [Электронный ресурс]. – Режим доступа: <https://acm.bsu.by/wiki/Unix2019b/>. – Дата доступа: 11.02.2024.

[2] Архитектура Unix. Процессы [Электронный ресурс]. – Режим доступа: http://heap.altlinux.org/tmp/unix\_base/ch01s03.html. – Дата доступа: 13.02.2024.

[3] Разделяемая память и семафоры [Электронный ресурс]. – Режим доступа: https://debianinstall.ru/razdelyaemaya-pamyat-semafory-i-ocheredi-soobshhenij-v-os-linux/. – Дата доступа: 13.02.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код lab4.c

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <semaphore.h>

#include <time.h>

#define NUM\_BLOCKS 5

#define BLOCK\_SIZE 1

typedef struct {

int intValue;

} DataBlock;

void create\_agents(key\_t key, int id[], DataBlock \*memory[], sem\_t \*sem[]) {

int i;

for (i = 0; i < NUM\_BLOCKS; ++i) {

pid\_t pid = fork();

if (pid == -1) {

perror("fork");

exit(1);

} else if (pid == 0) {

key\_t process\_key = key + i;

sem\_wait(sem[i]);

printf("\*\*\*\*\*\nProcess %d\n", getpid());

srand(time(NULL) + getpid() + i);

for (int j = 0; j < BLOCK\_SIZE; ++j) {

memory[i][j].intValue = rand() % 100;

printf("\n Shared memory ID: %d\n Key: %d\n Memory[%d][%d].intValue = %d\n",

id[i], process\_key, i, j, memory[i][j].intValue);

}

printf("\*\*\*\*\*");

sem\_post(sem[i]);

exit(0);

}

}

}

int main() {

int id[NUM\_BLOCKS];

key\_t key;

DataBlock \*memory[NUM\_BLOCKS];

sem\_t \*sem[NUM\_BLOCKS];

int i;

for (i = 0; i < NUM\_BLOCKS; ++i) {

char sem\_name[20];

sprintf(sem\_name, "/semaphore%d", i);

sem[i] = (sem\_t \*)malloc(sizeof(sem\_t));

if (sem[i] == NULL) {

perror("malloc");

exit(1);

}

if (sem\_init(sem[i], 1, 1) == -1) {

perror("sem\_init");

exit(1);

}

}

if ((key = ftok(".", 'R')) == -1) {

perror("ftok");

exit(1);

}

printf("\*\*\*\*\*\nMain Process PID: %d\n", getpid());

for (i = 0; i < NUM\_BLOCKS; ++i) {

key\_t process\_key = key + i;

if ((id[i] = shmget(process\_key, BLOCK\_SIZE \* sizeof(DataBlock), IPC\_CREAT | IPC\_EXCL | 0644)) == -1) {

perror("shmget");

exit(1);

}

printf("Shared memory for Block %d: ID: %d, Key: %d\n", i + 1, id[i], process\_key);

memory[i] = (DataBlock \*)shmat(id[i], NULL, 0);

if (memory[i] == (DataBlock \*) -1) {

perror("shmat");

exit(1);

}

}

for (i = 0; i < NUM\_BLOCKS; ++i) {

for (int j = 0; j < BLOCK\_SIZE; ++j) {

memory[i][j].intValue = 0;

}

}

create\_agents(key, id, memory, sem);

for (i = 0; i < NUM\_BLOCKS; ++i) {

wait(NULL);

}

int sum = 0;

for (i = 0; i < NUM\_BLOCKS; ++i) {

for (int j = 0; j < BLOCK\_SIZE; ++j) {

sum += memory[i][j].intValue;

}

}

printf("\nSUM: %d\n", sum);

for (i = 0; i < NUM\_BLOCKS; ++i) {

if (shmdt(memory[i]) == -1) {

perror("shmdt");

exit(1);

}

}

for (i = 0; i < NUM\_BLOCKS; ++i) {

if (shmctl(id[i], IPC\_RMID, NULL) == -1) {

perror("shmctl");

exit(1);

}

}

for (i = 0; i < NUM\_BLOCKS; ++i) {

if (sem\_destroy(sem[i]) == -1) {

perror("sem\_destroy");

}

free(sem[i]);

}

return 0;

}