МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системное программирование»

Лабораторная работа №8

**«Реализация обмена данными между процессами»**

по дисциплине

Операционные системы

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Королькова Т.В.

Москва, 2025 г

Написать программу, выполняющую с помощью ВСЕХ перечисленных

механизмов межпроцессного взаимодействия (разделяемая память, сокеты,

каналы) одну из задач (в соответствии с № по журналу), приведенных в Таблице

1. Выбор средства коммуникации в программе осуществлять исходя из

введенного пользователем при запуске программы аргумента, например:

./lab\_4 pipe

запускает ведущую программу, задавая режим коммуникации с помощью

каналов, далее ведущий процесс создает дочерние.

Для решения задачи создавать минимально 2 ведомых процесса,

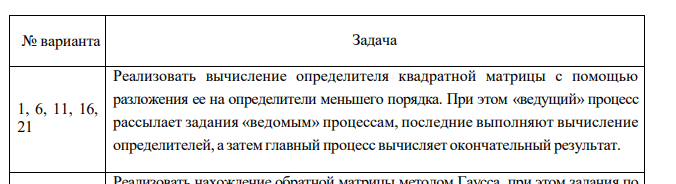
выполняющих переданные ведущим процессом подзадачи и возвращающие

результаты ведущему процессу. Финальную обработку результатов, по

9

необходимости, осуществлять ведущим процессом. Продумать интерфейс

взаимодействия с пользователем и отображение результатов работы



**ВЫПОЛНЕНИЕ**



Рисунок 1 – Запуск без аргумента

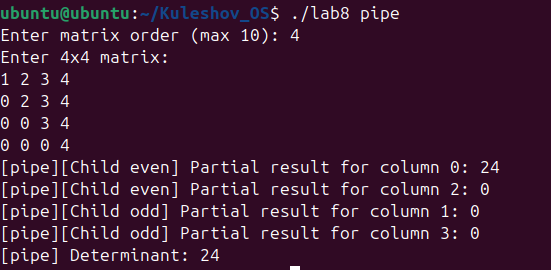


Рисунок 2 – Запуск с pipe

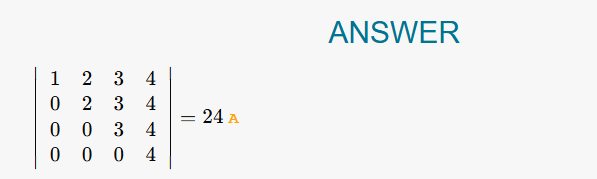


Рисунок 3 – Перепроверка

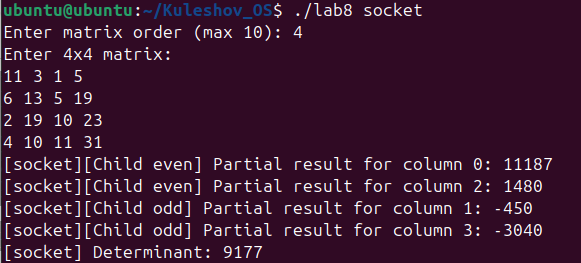


Рисунок 4 – Запуск с socket

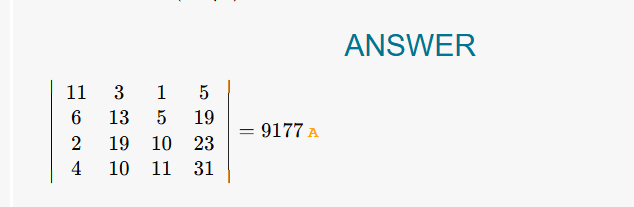


Рисунок 5 – Перепроверка

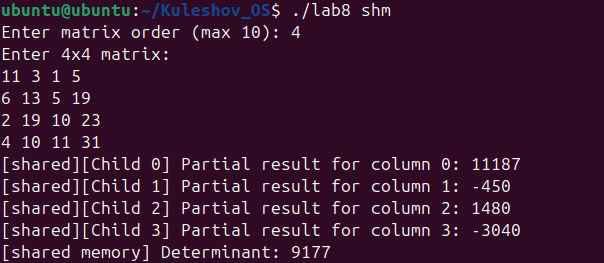


Рисунок 6 – Запуск с shared memory

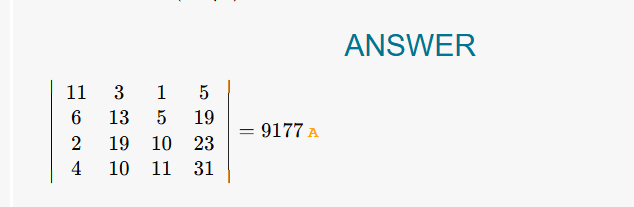


Рисунок 7 – Перепроверка

Код для задания

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#define MAX 10

#define PORT 8080

**void** **getMinor**(**int** src[MAX][MAX], **int** dest[MAX][MAX], **int** n, **int** row, **int** col) {

**int** r = **0**, c = **0**;

**for** (**int** i = **0**; i < n; i++) {

**if** (i == row) **continue**;

c = **0**;

**for** (**int** j = **0**; j < n; j++) {

**if** (j == col) **continue**;

dest[r][c++] = src[i][j];

}

r++;

}

}

**int** **determinant**(**int** mat[MAX][MAX], **int** n) {

**if** (n == **1**) **return** mat[**0**][**0**];

**if** (n == **2**) **return** mat[**0**][**0**]\*mat[**1**][**1**] - mat[**0**][**1**]\*mat[**1**][**0**];

**int** minor[MAX][MAX], det = **0**;

**for** (**int** c = **0**; c < n; c++) {

getMinor(mat, minor, n, **0**, c);

**int** sign = (c % **2** == **0**) ? **1** : -**1**;

det += sign \* mat[**0**][c] \* determinant(minor, n - **1**);

}

**return** det;

}

// PIPE

**void** **run\_pipe**(**int** matrix[MAX][MAX], **int** n) {

**int** pipe1[**2**][**2**], pipe2[**2**][**2**];

pipe(pipe1[**0**]); pipe(pipe1[**1**]);

pipe(pipe2[**0**]); pipe(pipe2[**1**]);

**if** (fork() == **0**) {

close(pipe1[**1**][**1**]); close(pipe1[**0**][**0**]);

**int** nn; **int** mat[MAX][MAX];

read(pipe1[**1**][**0**], &nn, **sizeof**(**int**));

read(pipe1[**1**][**0**], mat, **sizeof**(mat));

**int** sum = **0**, minor[MAX][MAX];

**for** (**int** i = **0**; i < nn; i++) {

**if** (i % **2** != **0**) **continue**;

getMinor(mat, minor, nn, **0**, i);

**int** sign = (i % **2** == **0**) ? **1** : -**1**;

**int** part = sign \* mat[**0**][i] \* determinant(minor, nn - **1**);

printf("[pipe][Child even] Partial result for column %d: %d**\n**", i, part);

sum += part;

}

write(pipe1[**0**][**1**], &sum, **sizeof**(**int**));

exit(**0**);

}

**if** (fork() == **0**) {

close(pipe2[**1**][**1**]); close(pipe2[**0**][**0**]);

**int** nn; **int** mat[MAX][MAX];

read(pipe2[**1**][**0**], &nn, **sizeof**(**int**));

read(pipe2[**1**][**0**], mat, **sizeof**(mat));

**int** sum = **0**, minor[MAX][MAX];

**for** (**int** i = **0**; i < nn; i++) {

**if** (i % **2** == **0**) **continue**;

getMinor(mat, minor, nn, **0**, i);

**int** sign = (i % **2** == **0**) ? **1** : -**1**;

**int** part = sign \* mat[**0**][i] \* determinant(minor, nn - **1**);

printf("[pipe][Child odd] Partial result for column %d: %d**\n**", i, part);

sum += part;

}

write(pipe2[**0**][**1**], &sum, **sizeof**(**int**));

exit(**0**);

}

close(pipe1[**1**][**0**]); close(pipe1[**0**][**1**]);

close(pipe2[**1**][**0**]); close(pipe2[**0**][**1**]);

write(pipe1[**1**][**1**], &n, **sizeof**(**int**));

write(pipe1[**1**][**1**], matrix, **sizeof**(**int**) \* MAX \* MAX);

write(pipe2[**1**][**1**], &n, **sizeof**(**int**));

write(pipe2[**1**][**1**], matrix, **sizeof**(**int**) \* MAX \* MAX);

**int** res1, res2;

read(pipe1[**0**][**0**], &res1, **sizeof**(**int**));

read(pipe2[**0**][**0**], &res2, **sizeof**(**int**));

wait(NULL); wait(NULL);

printf("[pipe] Determinant: %d**\n**", res1 + res2);

}

// SOCKET

**void** **client\_process\_socket**(**int** is\_even) {

sleep(**1**);

**int** sock = socket(AF\_INET, SOCK\_STREAM, **0**);

**struct** sockaddr\_in serv\_addr;

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

connect(sock, (**struct** sockaddr\*)&serv\_addr, **sizeof**(serv\_addr));

**int** n, matrix[MAX][MAX];

read(sock, &n, **sizeof**(**int**));

read(sock, matrix, **sizeof**(matrix));

**int** sum = **0**, minor[MAX][MAX];

**for** (**int** i = **0**; i < n; i++) {

**if** ((i % **2** == **0**) != is\_even) **continue**;

getMinor(matrix, minor, n, **0**, i);

**int** sign = (i % **2** == **0**) ? **1** : -**1**;

**int** part = sign \* matrix[**0**][i] \* determinant(minor, n - **1**);

printf("[socket][Child %s] Partial result for column %d: %d**\n**",

is\_even ? "even" : "odd", i, part);

sum += part;

}

write(sock, &sum, **sizeof**(**int**));

close(sock);

exit(**0**);

}

**void** **run\_socket**(**int** matrix[MAX][MAX], **int** n) {

**int** server\_fd = socket(AF\_INET, SOCK\_STREAM, **0**);

**struct** sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

addr.sin\_addr.s\_addr = INADDR\_ANY;

bind(server\_fd, (**struct** sockaddr\*)&addr, **sizeof**(addr));

listen(server\_fd, **2**);

**if** (fork() == **0**) client\_process\_socket(**1**);

**if** (fork() == **0**) client\_process\_socket(**0**);

**int** res[**2**];

**for** (**int** i = **0**; i < **2**; i++) {

**int** client = accept(server\_fd, NULL, NULL);

write(client, &n, **sizeof**(**int**));

write(client, matrix, **sizeof**(**int**) \* MAX \* MAX);

read(client, &res[i], **sizeof**(**int**));

close(client);

}

wait(NULL); wait(NULL);

close(server\_fd);

printf("[socket] Determinant: %d**\n**", res[**0**] + res[**1**]);

}

// SHARED MEMORY

**void** **run\_shared**(**int** matrix[MAX][MAX], **int** n) {

**key\_t** key = IPC\_PRIVATE;

**int** shmid = shmget(key, **sizeof**(**int**) \* n, IPC\_CREAT | **0666**);

**int**\* results = (**int**\*) shmat(shmid, NULL, **0**);

**for** (**int** i = **0**; i < n; i++) {

**if** (fork() == **0**) {

**int** minor[MAX][MAX];

getMinor(matrix, minor, n, **0**, i);

**int** sign = (i % **2** == **0**) ? **1** : -**1**;

results[i] = sign \* matrix[**0**][i] \* determinant(minor, n - **1**);

printf("[shared][Child %d] Partial result for column %d: %d**\n**", i, i, results[i]);

shmdt(results);

exit(**0**);

}

}

**for** (**int** i = **0**; i < n; i++) wait(NULL);

**int** final\_det = **0**;

**for** (**int** i = **0**; i < n; i++) final\_det += results[i];

printf("[shared memory] Determinant: %d**\n**", final\_det);

shmdt(results);

shmctl(shmid, IPC\_RMID, NULL);

}

// MAIN

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

**if** (argc < **2**) {

printf("Usage: %s [pipe|socket|shm]**\n**", argv[**0**]);

**return** **1**;

}

**int** n, matrix[MAX][MAX];

printf("Enter matrix order (max %d): ", MAX);

scanf("%d", &n);

printf("Enter %dx%d matrix:**\n**", n, n);

**for** (**int** i = **0**; i < n; i++)

**for** (**int** j = **0**; j < n; j++)

scanf("%d", &matrix[i][j]);

**if** (strcmp(argv[**1**], "pipe") == **0**)

run\_pipe(matrix, n);

**else** **if** (strcmp(argv[**1**], "socket") == **0**)

run\_socket(matrix, n);

**else** **if** (strcmp(argv[**1**], "shm") == **0** || strcmp(argv[**1**], "shared") == **0**)

run\_shared(matrix, n);

**else** {

printf("Unknown method: %s**\n**", argv[**1**]);

**return** **1**;

}

**return** **0**;

}