Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине «Основы операционных систем»

на тему «Работа с разделяемой памятью»

Выполнили студенты группы 22ВВП1

Демин М. С.

Беляев Д. И.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Название**

Работа с разделяемой памятью

**Цель работы**

Изучение способов работы с разделяемой памятью на основе файлов,

проецируемых в память. Изучение способов синхронизации процессов и

нитей с использованием мьютексов (mutex) и семафоров.

**Лабораторное задание**

Имеется 3 процесса. Два из них производят инкремент переменной в

разделяемой памяти на 1. Одна – декремент на 2. Работа с переменной

должна осуществляться с помощью критической секции на основе

мьютексов. Вывести на экран текущее значение переменной и

крайние значения.

**Описание данных**

1. Константы:

- SHARED\_MEMORY\_NAME: Имя общей памяти, используемой для межпроцессного взаимодействия.

- MUTEX\_NAME: Имя мьютекса, используемого для синхронизации доступа к общей памяти.

- INITIAL\_VALUE: Начальное значение, которое будет записано в общую память (0).

- PROCESS\_COUNT: Количество процессов, которые будут созданы (3).

2. Функции:

- Increment(int val): Увеличивает значение в общей памяти на val (1). Запускается 10 раз с паузой в 100 мс между итерациями.

- Decrement(int val): Уменьшает значение в общей памяти на val (2). Также запускается 10 раз с паузой в 100 мс между итерациями.

3. Функция CreateStartPath(char argv):

- Создает строку, содержащую путь к исполняемому файлу и аргумент командной строки ('i' или 'd').

4. Основная функция main(int argc, char\* argv[]):

- Проверяет аргументы командной строки: если передан аргумент 'i', запускает функцию Increment, если 'd' — функцию Decrement.

- Создает общую память и инициализирует ее начальным значением.

- Создает мьютекс для синхронизации.

- Запускает несколько процессов (в зависимости от значения PROCESS\_COUNT), каждый из которых будет либо увеличивать, либо уменьшать значение в общей памяти.

- Ждет завершения всех дочерних процессов и выводит финальное значение.

**Описание структуры программы**

1. Инициализация:

- Программа начинается с проверки аргументов командной строки. Если аргументы не переданы, она создает общую память и мьютекс.

2. Создание процессов:

- В цикле создаются три процесса, каждый из которых выполняет либо инкремент, либо декремент значения в общей памяти. Аргумент определяется по индексу (четные — инкременты, нечетные — декременты).

3. Синхронизация:

- Для предотвращения гонок данных используется мьютекс. Каждый процесс перед изменением значения в общей памяти должен захватить мьютекс, что гарантирует, что только один процесс может изменять значение в данный момент времени.

4. Завершение:

- После завершения всех дочерних процессов основная программа выводит финальное значение из общей памяти, освобождает ресурсы (освобождает общую память и закрывает дескрипторы) и завершает выполнение.

**Текст программы**

#include <windows.h>

#include <stdio.h>

#define SHARED\_MEMORY\_NAME L"SharedMemory"

#define MUTEX\_NAME L"SharedMemoryMutex"

#define INITIAL\_VALUE 0

#define PROCESS\_COUNT 3

void Increment(int val)

{

HANDLE hMapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, SHARED\_MEMORY\_NAME);

int\* pSharedMemory = (int\*)MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, MUTEX\_NAME);

for (int i = 0; i < 10; ++i) {

WaitForSingleObject(hMutex, INFINITE);

(\*pSharedMemory) += val;

wprintf(L"Increment: %d\n", \*pSharedMemory);

ReleaseMutex(hMutex);

Sleep(100);

}

UnmapViewOfFile(pSharedMemory);

CloseHandle(hMapFile);

CloseHandle(hMutex);

}

void Decrement(int val)

{

HANDLE hMapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, SHARED\_MEMORY\_NAME);

int\* pSharedMemory = (int\*)MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, MUTEX\_NAME);

for (int i = 0; i < 10; ++i) {

WaitForSingleObject(hMutex, INFINITE);

(\*pSharedMemory) -= val;

wprintf(L"Decrement: %d\n", \*pSharedMemory);

ReleaseMutex(hMutex);

Sleep(100);

}

UnmapViewOfFile(pSharedMemory);

CloseHandle(hMapFile);

CloseHandle(hMutex);

}

LPWSTR CreateStartPath(char argv)

{

WCHAR path[MAX\_PATH];

GetModuleFileName(NULL, path, sizeof(path) / sizeof(path[0]));

LPWSTR pathArgs = (LPWSTR)malloc(MAX\_PATH \* sizeof(WCHAR));

swprintf(pathArgs, MAX\_PATH, L"%s %c", path, argv);

return pathArgs;

}

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

{

if (argc > 1) {

if (argv[1][0] == 'i') {

Increment(1);

}

else if (argv[1][0] == 'd') {

Decrement(2);

}

return 0;

}

HANDLE hMapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, sizeof(int), SHARED\_MEMORY\_NAME);

int\* pSharedMemory = (int\*)MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(int));

\*pSharedMemory = INITIAL\_VALUE;

HANDLE hMutex = CreateMutex(NULL, FALSE, MUTEX\_NAME);

PROCESS\_INFORMATION pi[PROCESS\_COUNT] = { 0 };

for (size\_t i = 0; i < PROCESS\_COUNT; i++) {

STARTUPINFO si = { sizeof(si) };

LPWSTR startPath = CreateStartPath(i % 2 == 0 ? 'i' : 'd');

CreateProcess(NULL, startPath, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi[i]);

free(startPath);

}

HANDLE handles[PROCESS\_COUNT] = { 0 };

for (size\_t i = 0; i < PROCESS\_COUNT; i++) {

handles[i] = pi[i].hProcess;

}

WaitForMultipleObjects(3, handles, TRUE, INFINITE);

wprintf(L"Final value: %d\n", \*pSharedMemory);

UnmapViewOfFile(pSharedMemory);

CloseHandle(hMapFile);

CloseHandle(hMutex);

for (int i = 0; i < PROCESS\_COUNT; ++i) {

CloseHandle(pi[i].hProcess);

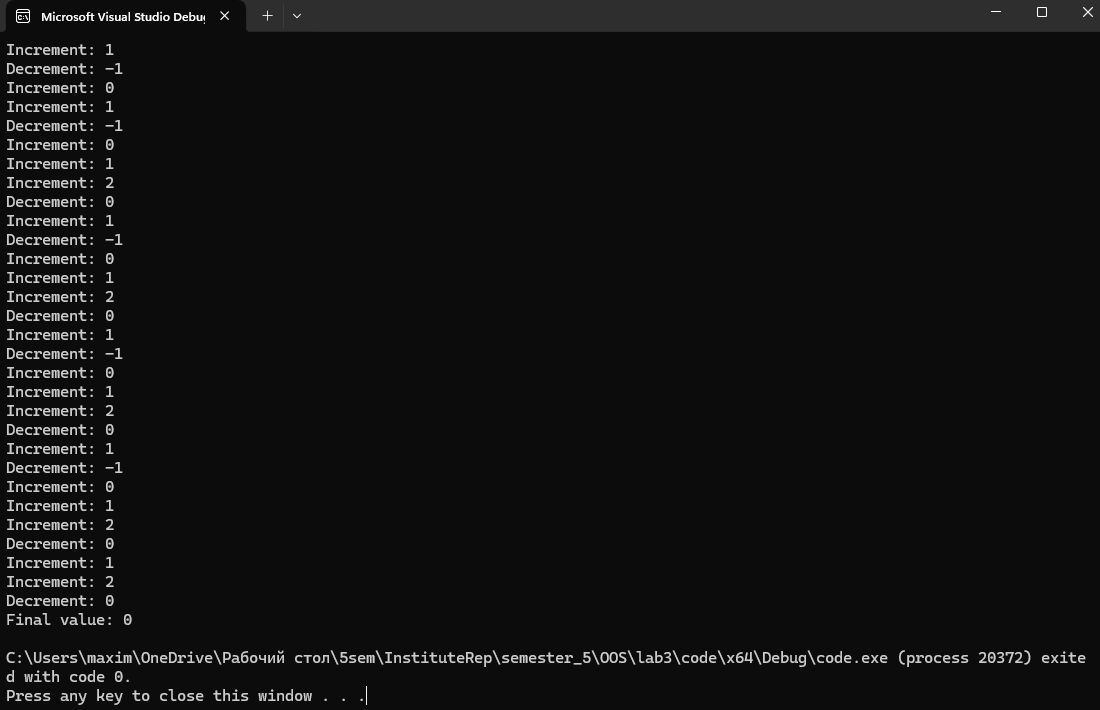
CloseHandle(pi[i].hThread);

}

return 0;

}

**Результат работы программы**



**Вывод**

Изучили способы работы с разделяемой памятью на основе файлов,

проецируемых в память. Изучили способы синхронизации процессов и

нитей с использованием мьютексов (mutex) и семафоров.