МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 11

На тему: *“Організація взаємодії між процесами ”* ***З дисципліни:***
“Операційні системи”

Лектор:
ст. вик. ПЗ
Грицай О.Д.

Виконала:
ст. гр. ПЗ-26
Галамай В.С.

Прийняв:
доц. каф. ПЗ
Горечко О.М.

« ____ » _____ 2021 р.
∑= ____ _ _ _ _ _ _ _ _ _ _ _

Львів – 2021

Тема. Організація взаємодії між процесами

Мета. Ознайомитися зі способами міжпроцесної взаємодії. Ознайомитися з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчитися працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу.

Теоретичні відомості

Міжпроцесовий зв’язок із використанням спільної пам'яті вимагає комунікаційних процесів для встановлення області спільної пам'яті. Зазвичай область спільної пам’яті міститься в адресному просторі процесу, створюючи сегмент спільної пам’яті. Як правило, операційна система намагається запобігти доступу одного процесу до пам’яті іншого процесу. Загальна пам’ять вимагає, щоб два більше обробляли процес, щоб зменшити це обмеження. Потім вони можуть обмінюватися інформацією, читаючи та записуючи спільні ділянки. Форма даних та місцеположення визначаються цими процесами і не підконтрольні операційній системі. Процеси також відповідають за те, щоб вони не записували в одне місце одночасно. Щоб проілюструвати концепцію процесів співпраці, розглянемо проблему виробник-споживач, яка є звичайною парадигмою для співпрацюючих процесів. Процес виробника виробляє інформацію, яку споживає споживчий процес. Щоб дозволити процесам виробника та споживача одночасно працювати, ми повинні мати доступний буфер елементів, який може бути заповнений виробником і спорожнений споживачем. Цей буфер розміщуватиметься в області пам’яті, яка поділяється між виробництвом та споживчими процесами. Виробник може вироблятись між тим, як споживатиме консолідований інший спосіб. Виробник і споживач повинні бути синхронізовані, щоб споживач не намагався споживати товар, який ще не був вироблений. Можна використовувати два типи буферів. Незв'язаний буфер не обмежує розмір буфера. Споживачеві, можливо, доведеться чекати нових товарів, але виробник завжди може випускати нові товари. Обмежений буфер передбачає фіксований розмір буфера. У цьому випадку споживач повинен чекати, якщо буфер порожній, а виробник повинен чекати, якщо буфер заповнений.

File Mapping

Відображення файлів дозволяє процесу обробляти вміст файлу так, ніби вони є блоком пам'яті в адресному просторі процесу. Процес може використовувати прості операції з покажчиком для вивчення та зміни вмісту файлу. Коли два або більше процесів отримують доступ до одного і того ж картографічного відображення, кожен процес отримує вказівник на пам'ять у власному адресному просторі, який він може використовувати для читання або зміни вмісту файлу. Процеси повинні використовувати об'єкт синхронізації, такий як семафор, для запобігання пошкодження даних у багатозадачному середовищі. Ви можете використовувати спеціальний випадок відображення файлів для забезпечення спільної пам’яті між процесами. Якщо при створенні об'єкта відображення файлів ви вказуєте системний файл заміни, об'єкт файлового відображення розглядається як блок спільної пам'яті. Інші процеси можуть отримати доступ до одного і того ж блоку пам'яті, відкривши той самий об’єкт відображення файлів. Відображення файлів є досить ефективним, а також забезпечує атрибути безпеки, підтримувані операційною системою, що може запобігти несанкціонованому пошкодженню даних. Відображення файлів можна використовувати лише між процесами на локальному комп'ютері; його не можна використовувати через мережу. Відображення файлів є ефективним способом обміну даними на двох або більше процесах на одному комп’ютері, але ви повинні забезпечити синхронізацію між процесами.

Завдання

1. Реалізувати алгоритм моделювання заданої задачі за допомогою окремих процесів згідно індивідуального завдання.
2. Реалізувати синхронізацію роботи процесів.
3. Забезпечити зберігання результатів виконання завдання.
4. Результати виконання роботи відобразити у звіті.

Варіант-3 Створити програму, що моделює наступну ситуацію: Процес-науковий керівник проекту пропонує виконавців проекту-дочірні процеси. Процес-керівник створює додаток-віртуальну дошку (файл), де можна генерувати ідеї для проекту. Процеси-виконавці генерують ідеї, записуючи їх на спільну дошку. На виконання даного завдання вони мають 3 хвилини, після чого процес-керівник призупиняє їхню роботу і виводить на екран усі згенеровані ідеї, нумеруючи кожну з них. Процеси-виконавці голосують за три найкращі ідеї. Після чого процес-керівник записує на

дошку три найкращі ідеї і закриває роботу додатку-віртуальної дошки, зберігаючи її вміст. Реалізувати дану модель, використовуючи **а) файли, що проектуються у пам'ять;** пайпи (робота в межах однієї системи)

Виконання завдання командою:

Боб Олександр - розробляє частину головного процесу, який створює дошку (файл) і запускає інші процеси

Вишневська Соломія - розробляє процеси-виконавці, що генерують ідеї та записують на спільну дошку

Галамай Віта - розробляє голосування процесів-виконавців та готує звіт про виконання завдання

Томчишин Вікторія - розробляє вивід результатів голосування на дошку, готує презентацію виконаного завдання

Хід роботи

1. Код програми:

(Написаний мною код виділений жирним шрифтом)

mainTeacher.cpp:

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <Windows.h>
#include <tchar.h>
#include <string>

using namespace std;

#define SIZE 9000
TCHAR name[] = TEXT("BOARD");
TCHAR name2[] = TEXT("VOTING");

class Student
{
public:
     STARTUPINFOA si;
     PROCESS_INFORMATION pi;
};
```

```
HANDLE mapping;
HANDLE mapping2;
HANDLE boardFile;
HANDLE voteFile;
HANDLE accessFile;
HANDLE timeMUT;
HANDLE accessToBoardMUT;
Student* students = NULL;


int main(int argc, char* argv[])
{
	cout << "Hello, i'm teacher" << endl;

	boardFile = CreateFileA("..\\..\\board.txt", GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
	voteFile = CreateFileA("..\\..\\voting.txt", GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

	mapping = CreateFileMapping(boardFile, NULL, PAGE_READWRITE,
0, SIZE, name);
	if (mapping == nullptr) { return -1; }

	mapping2 = CreateFileMapping(voteFile, NULL, PAGE_READWRITE,
0, SIZE, name2);
	if (mapping2 == nullptr) { return -1; }

	cout << "Enter count of students: " << endl;
	int countOfStudents;
	cin >> countOfStudents;

	if (countOfStudents > 99) {
		cout << "Too big number of students.\nMaximum number is
99\n";
		return -1;
	}

	students = new Student[countOfStudents];
	for (int i = 0; i < countOfStudents; ++i) {
		ZeroMemory(&(students[i].si), sizeof(STARTUPINFO));
		students[i].si.cb = sizeof(STARTUPINFO);
		ZeroMemory(&(students[i].pi),
sizeof(PROCESS_INFORMATION));
	}
```

```
	string studentFilePath =
"..\\..\\StudentProcess\\Debug\\StudentProcess.exe ";

	accessFile = CreateFileA("..\\..\\access.txt", GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
	DWORD wr;
	char* access = new char[2];
	access[0] = '1';
	access[1] = '\0';
	WriteFile(accessFile, access, 2, &wr, NULL);
	CloseHandle(accessFile);



	for (int i = 0; i < countOfStudents; i++)
	{
		string newStudentFilePath = studentFilePath;

		//number of student
		int k = i + 1;
		string r = std::to_string(k);
		newStudentFilePath.push_back(r[0]);
		if (k > 9) {
			newStudentFilePath.push_back(r[1]);
		}
		LPSTR studentFilePathLPSTR =
const_cast<char*>((newStudentFilePath).c_str());

		//creating process
		CreateProcessA(NULL, studentFilePathLPSTR, NULL, NULL,
true, CREATE_NEW_CONSOLE, NULL, NULL, &(students[i].si),
&(students[i].pi));
	}


	 Sleep(2 * 60 * 1000);//2 minutes wait

	timeMUT = CreateMutexW(NULL, false, (LPCWSTR)"timeMUT");
	WaitForSingleObject(timeMUT, INFINITE);
	accessFile = CreateFileA("..\\..\\access.txt", GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
	access[0] = '0';
	access[1] = '\0';
	WriteFile(accessFile, access, 2, &wr, NULL);
	CloseHandle(accessFile);
	ReleaseMutex(timeMUT);
```

```
	//kill processes
	for (int i = 0; i < countOfStudents; i++) {
		TerminateProcess(&(students[i].pi.hProcess), 0);
	}

	cout << "Time's over.\n\nWait for loading ideas...\n\n";

	Sleep(1 * 60 * 1500);//1 minute wait

	//looking for the end
	LPVOID pBuf = (void*)MapViewOfFile(mapping, FILE_MAP_READ, 0,
0, SIZE);
	if (pBuf == nullptr) { return -1; }
	char* newArr = (char*)pBuf;
	int lastn = 0;
	for (int i = 0; i < SIZE; i++) {
		if (newArr[i] == '\n') {
			lastn = i;
		}
	}

	//reading file, write to console
	newArr = (char*)pBuf;
	int k = 0;
	cout << "Ideas from the board:\n" << k + 1 << ".\t";
	k++;
	for (int i = 0; i < lastn + 1; i++) {
		if (i > 0 && newArr[i - 1] == '\n') {
			cout << k + 1 << ".\t";
			k++;
		}
		cout << newArr[i];
	}



	cout << "\nWait for the results of voting...\n\n";

    Sleep(1 * 60 * 1500);

	//looking for the end
	LPVOID pBuf2 = (void*)MapViewOfFile(mapping2, FILE_MAP_READ,
0, 0, SIZE);
	if (pBuf2 == nullptr) { return -1; }
	char* newArr2 = (char*)pBuf2;
	int* nums = new int[k+1];
	for (int i = 0; i < k + 1; i++) {
		nums[i] = 0;
```

```
	}

	int lastn2 = 0;
	for (int i = 0; i < SIZE; i++) {
		if (newArr2[i] == '\n') {
			lastn2 = i;
		}
	}

	for (int i = 0; i < lastn2; i++) {
		for (int j = 0; j <= k; j++) {
			int l = j + 1;
			if (newArr2[i] == to_string(l)[0]) {
				nums[l]++;
			}
		}
	}
	//for (int i = 0; i < lastn2; i++) {
	//	cout << "Idea: "<< i << " Votes: " << nums[i] << endl;
	//}

	//selecting top 3

	//find max quontity of votes for one idea
	int maxVotes = 0;
	for (int i = 0; i < lastn2; i++) {
		if (maxVotes < nums[i]) maxVotes = nums[i];
	}

	//find indexes of top-3 ideas
	int topIndexes[3] = {0,0,0};
	int currentPosition = 0;
	while (currentPosition <= 3 && maxVotes>0) {
		for (int i = 0; i < lastn2; i++) {
			if (maxVotes == nums[i]) {
				topIndexes[currentPosition] = i;
				currentPosition++;
			}
		}
		maxVotes--;
	}

	cout << "Top-3:\n" << endl;

		//looking for the end
	 pBuf = (void*)MapViewOfFile(mapping, FILE_MAP_READ, 0, 0,
SIZE);
	if (pBuf == nullptr) { return -1; }
```

```
 newArr = (char*)pBuf;
lastn = 0;
for (int i = 0; i < SIZE; i++) {
     if (newArr[i] == '\n') {
          lastn = i;
     }
}


//reading file, write to console
newArr = (char*)pBuf;
string result = "\nTop-3:\n";
for (int p = 0; p < 3; p++) {
     k = 1;
     bool boolka = false;
     for (int i = 0; i < lastn + 1; i++) {
          if (boolka == false) {
               if (i > 0 && newArr[i - 1] == '\n') {
                    k++;
                    if (k == topIndexes[p]) {
                         cout << k << ". ";
                         cout << newArr[i];
                         result += to_string(k)+ ". ";
                         result += newArr[i];
                         boolka = true;
                    }
               }

          }
          else if (boolka) {
               cout << newArr[i];
               result += newArr[i];
               if (newArr[i + 1] == '\n') {
                    cout << endl;
                    result += "\n";
                    boolka = false;
               }
          }
     }
}

//cout << result;
//open mapping
char* charPointerV = NULL;
HANDLE mappingV = OpenFileMapping(FILE_MAP_ALL_ACCESS, true,
name);
if (mappingV == NULL) { return -1; }
```

```
	LPVOID pBufV = (void*)MapViewOfFile(mappingV,
FILE_MAP_ALL_ACCESS, 0, 0, SIZE);
	if (pBufV == nullptr) { return -1; }

	//set pointer in the end of file
	newArr = (char*)pBufV;
	lastn = 0;
	for (int i = 0; i < SIZE; i++) {
		if (newArr[i] == '\n') {
			lastn = i;
		}
	}
	charPointerV = (char*)pBufV;
	charPointerV += lastn + 1;
	pBufV = (void*)charPointerV;


	//write to file
	snprintf((char*)pBufV, sizeof(char)* result.length(),
result.c_str());

	getchar();
	getchar();
	getchar();

	UnmapViewOfFile(mapping);
	UnmapViewOfFile(mapping2);
	CloseHandle(boardFile);
	CloseHandle(voteFile);

	return 0;
}
```

mainStudent.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <Windows.h>
#include <tchar.h>
#include <string>
#include <cstring>

using namespace std;

#define SIZE 9000
TCHAR name[] = TEXT("BOARD");
```

```
TCHAR nameV[] = TEXT("VOTING");
HANDLE accessFile;
HANDLE timeMUT;
HANDLE mutBoard;
HANDLE mutVote;
HANDLE mapping;
HANDLE mappingV;



int main(int argc, char* argv[])
{
	//number of student
	int studentNum = atoi(argv[1]);
	cout << "i`m student #" << studentNum << ".\n";

	//info to write in file
	char Data[30+sizeof(char)];
	std::sprintf(Data, "student # %d. Idea# ", studentNum);

	timeMUT = CreateMutexW(NULL, false, (LPCWSTR)"timeMUT");
	mutBoard = CreateMutexW(NULL, false, (LPCWSTR)"boardMUT");
	mutVote = CreateMutexW(NULL, false, (LPCWSTR)"voteMUT");

	char* charPointer = NULL;
	char* charPointerV = NULL;
	char* access = new char[2];
	DWORD rd;
	int numOfIdea = 0;


	while (true)
	{
		WaitForSingleObject(timeMUT, INFINITE);
		accessFile = CreateFileA("..\\..\\access.txt",
GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
		ReadFile(accessFile, access, 2, &rd, NULL);
		CloseHandle(accessFile);
		ReleaseMutex(timeMUT);
		if (access[0] == '0')
		{
			cout << "Time's over." << endl;
			break;
		}
		else
		{
			WaitForSingleObject(mutBoard, INFINITE);
```

```
                //open mapping
                mapping = OpenFileMapping(FILE_MAP_ALL_ACCESS,
true, name);
                if (mapping == NULL) { return -1; }
                LPVOID pBuf = (void*)MapViewOfFile(mapping,
FILE_MAP_ALL_ACCESS, 0, 0, SIZE);
                if (pBuf == nullptr) { return -1; }

                //set pointer in the end of file
                char* newArr = (char*)pBuf;
                int lastn = 0;
                for (int i = 0; i < SIZE; i++) {
                     if (newArr[i] == '\n') {
                          lastn = i;
                     }
                }
                charPointer = (char*)pBuf;
                charPointer += lastn + 1;
                pBuf = (void*)charPointer;

                //add extra info to write in file
                char newData[sizeof(Data)];
                strcpy(newData, Data);
                char ideas[3 + sizeof(char)];
                numOfIdea++;
                std::sprintf(ideas, "%d\n", numOfIdea);
                strcat(newData, ideas);

                //write to file
                snprintf((char*)pBuf, sizeof(newData), newData);
                cout << newData;

                Sleep(20000);
                ReleaseMutex(mutBoard);
           }
     }

     //voting
     //open mapping
     LPVOID pBuf2 = (void*)MapViewOfFile(mapping,
FILE_MAP_ALL_ACCESS, 0, 0, SIZE);
     if (pBuf2 == nullptr) { return -1; }

     //set pointer in the end of file
     char* newArr2 = (char*)pBuf2;
     int line = 0;
     for (int i = 0; i < SIZE; i++) {
```

```
            if (newArr2[i] == '\n') {
                  line++;
            }
      }

      int random =  rand() % line + 1;

      char DataV[4];
      std::sprintf(DataV, "%d\n", random);
      Sleep(20000);

      WaitForSingleObject(mutVote, INFINITE);

      //open mapping
      mappingV = OpenFileMapping(FILE_MAP_ALL_ACCESS, true, nameV);
      if (mappingV == NULL) { return -1; }
      LPVOID pBufV = (void*)MapViewOfFile(mappingV,
FILE_MAP_ALL_ACCESS, 0, 0, SIZE);
      if (pBufV == nullptr) { return -1; }

      //set pointer in the end of file
      char* newArr = (char*)pBufV;
      int lastn = 0;
      for (int i = 0; i < SIZE; i++) {
            if (newArr[i] == '\n') {
                  lastn = i;
            }
      }
      charPointerV = (char*)pBufV;
      charPointerV += lastn + 1;
      pBufV = (void*)charPointerV;

      //write to file
      snprintf((char*)pBufV, sizeof(DataV), DataV);
      cout << DataV;

      ReleaseMutex(mutVote);

      //getchar();
      UnmapViewOfFile(mapping);
      UnmapViewOfFile(mappingV);
      CloseHandle(mutBoard);
      CloseHandle(mutVote);

      return 0;
}
```

2. Реалізоване завдання

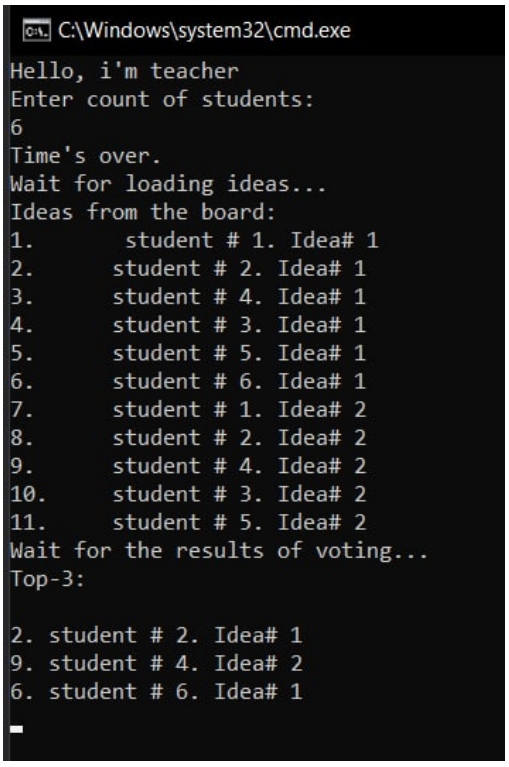

Рис.1 Результат виконання завдання у консолі

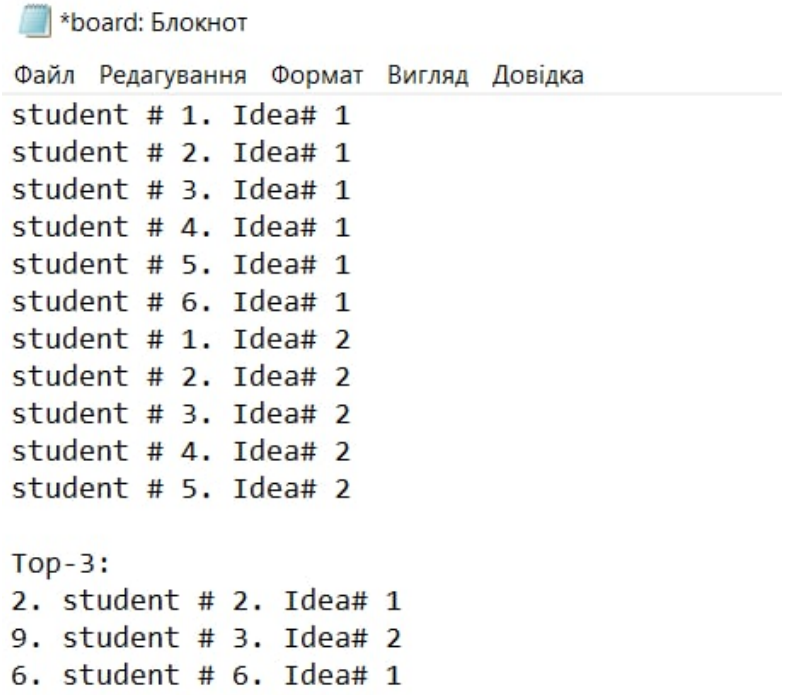

Рис.2 Результат виконання завдання на дошці(у файлі)

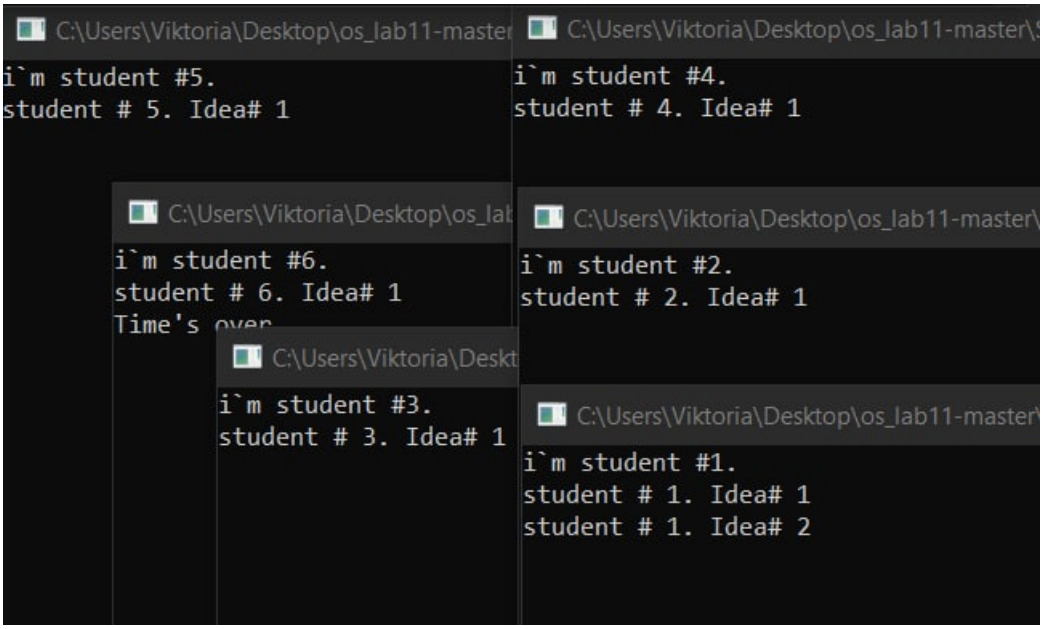

Рис.3 Створені процеси-виконавці

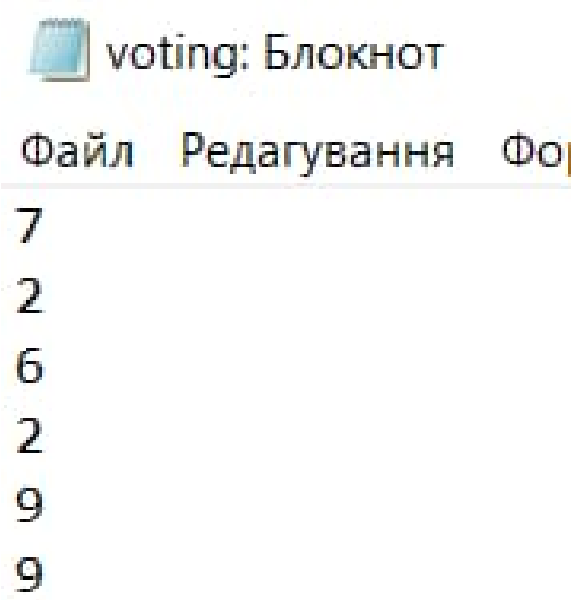

Рис.4 Дані про голосування, записані у файл

Висновок

На цій лабораторній роботі наша команда ознайомилася зі способами міжпроцесної взаємодії, з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчилася працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу. Боб Олександр - розробив частину головного процесу, який створює дошку (файл) і запускає інші

процеси. Вишневська Соломія - розробила процеси-виконавці, що генерують ідеї та записують на спільну дошку. Галамай Віта - розробила голосування процесів-виконавців. Томчишин Вікторія - розробила вивід результатів голосування на дошку.