Лабораторна робота № 1

Тема: Керування процесами та потоками.

Мета: Навчитися керувати процесами і потоками в середовищі операційної системи, розробляти програми керування процесами і потоками.

Обладнання та програмне забезпечення: ПК, операційна система Windows, середовище розробки Visual Studio.

Вказівки для самостійної підготовки

Під час підготовки необхідно повторити теоретичний матеріал:

1. Складові елементи процесу. Структури даних процесу.
2. Створення процесів. Завершення процесів.
3. Створення процесів у Win API. Завершення процесів у Win API.
4. Складові елементи потоку. Структури даних потоку.
5. Створення потоків у Win API. Завершення потоків у Win API.

Теоретичні відомості

1 Складові елементи процесу:

1. Адресний простір процесу складається з набору адрес віртуальної пам'яті, які він може використати. Адресний простір процесу недоступний іншим процесам.
2. Процес володіє системними ресурсами, такими як файли, мережні з'єднання, пристрої введення-виведення, об'єкти синхронізації тощо.
3. Процес містить деяку стартову інформацію для потоків, які в ньому створюватимуться.
4. Процес має містити хоча б один потік, який система скеровує на виконання. Без потоків у Windows наявність процесів неможлива.

2 Структури даних процесу

1. Для виконавчої системи Windows кожний процес зображується об'єктом-процесом виконавчої системи (executive process object); його також називають ***керуючим блоком процесу*** (executive process block, EPROCESS).
2. Для ядра системи процес зображується об'єктом-процесом ядра (kernel process object), його також називають ***блоком процесу ядра*** (process kernel block, KPROCESS).
3. У режимі користувача доступним є ***блок оточення процесу*** (process environment block, PEB), що перебуває в адресному просторі цього процесу.

Керуючий блок процесу містить такі основні елементи:

1. блок процесу ядра (KPROCESS);
2. ідентифікаційну інформацію;
3. інформацію про адресний простір процесу;

4. інформацію про ресурси, доступні процесу, та обмеження на використання цих ресурсів;
5. блок оточення процесу (PEB);
6. інформацію для підсистеми безпеки.

До ідентифікаційної інформації належать:

1. ідентифікатор процесу (pid);
2. ідентифікатор процесу, що створив цей процес;
3. ім'я завантаженого програмного файлу.

Блок процесу ядра містить усю інформацію, що належить до потоків цього процесу:

1. покажчик на ланцюжок блоків потоків ядра, де кожний блок відповідає потоку;
2. базову інформацію, необхідну ядру системи для планування потоків

Блок оточення процесу містить інформацію про процес, яка призначена для доступу з режиму користувача:

1. початкову адресу ділянки пам'яті, куди завантажився програмний файл;
2. покажчик на динамічну ділянку пам'яті, доступну процесу.

3 Створення процесів у Win API

Створення нового процесу виконується функцією CreateProcess().

```
BOOL CreateProcess(
  [in, optional]      LPCSTR                lpApplicationName,
  [in, out, optional] LPSTR                 lpCommandLine,
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  [in]                BOOL                  bInheritHandles,
  [in]                DWORD                 dwCreationFlags,
  [in, optional]      LPVOID                lpEnvironment,
  [in, optional]      LPCSTR                lpCurrentDirectory,
  [in]                LPSTARTUPINFOA        lpStartupInfo,
  [out]               LPPROCESS_INFORMATION lpProcessInformation
);
```

`lpApplicationName` —шлях до виконуваного файлу;
`lpCommandLine` - повний командний рядок для запуску виконуваного файлу;
`lpProcessAttributes`, `lpThreadAttributes` — атрибути безпеки для всього процесу і для головного потоку;
`bInheritHandles` - керує спадкуванням нащадками дескрипторів об'єктів.
`dwCreationFlags` - маска прапорів створенням нового процесу;

`lpEnvironment` - покажчик на пам'ять із новими змінними оточення, які предок може задаватидля нащадка;

`lpCurrentDirectory` - рядок із новим значенням поточного каталогу для нащадка;

`lpStartupInfo` - покажчик на структуру даних типу STARTUPINFO, на базі якої задають параметри для процесу-нащадка;

`lpProcessInformation` — покажчик на структуру даних PR0CESS_ INFORMATION, якузаповнює ОС під час виклику CreateProcess().

Структура PR0CESS_INF0RMATI0N містить чотири поля:

```
typedef struct _PROCESS_INFORMATION {
  HANDLE hProcess;
  HANDLE hThread;
  DWORD  dwProcessId;
  DWORD  dwThreadId;
} PROCESS_INFORMATION, *PPROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

`hProcess` — дескриптор створеного процесу;

`hThread` — дескриптор його головного потоку;

`dwProcessId` — ідентифікатор процесу (process id, pid);

`dwThreadId` — ідентифікатор головного потоку (thread id, tid).

Приклад виклику CreateProcess():

```
//... задається si
PROCESS_INFORMATION
pi;
    CreateProcess (NULL, " C:/winnt/notepad test.txt", NULL, NULL, TRUE,
        CREATE_NEW_CONSOLE, NULL, “D:/”, &si, &pi);
    printf ("pid=%d, t1d=%d\n", pi.dwProcessId, pi.dwThreadId);
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
```

Отримати дескриптор поточного процесу визначається функцієюGetCurrentProcess():

```
HANDLE curph = GetCurrentProcess();
```

Ідентифікатор поточного процесу визначається функцією GetCurrentProcessId():

```
int pid = GetCurrentProcessId();
```

4 Завершення процесів у Win API

Завершення процесів виконує функція ExitProcess():

```
void ExitProcess(
  [in] UINT uExitCode
);
```

uExitCode - код повернення процесу. Наприклад:

```
ExitProcess (100); // вихід з кодом 100
```

Завершення іншого процесу виконує функція TerminateProcess():

```
BOOL TerminateProcess(
  [in] HANDLE hProcess,
  [in] UINT   uExitCode
);
```

У процесі, який запустив інший процес, можна отримати код завершення цього процесу за допомогою функції

```
BOOL GetExitCodeProcess(
  [in]  HANDLE  hProcess,
  [out] LPDWORD lpExitCode
);
```

lpExitCode — покажчик на змінну, в яку заносять код завершення.

```
int exitcode;
GetExi tCodeProcess(pi.hProcess, &exitcode);
printf ("Код повернення =%d\n", exitcode);
```

5 Складові елементи потоку:

1. вміст набору регістрів, який визначає стан процесора;
2. два стеки — один використовують для роботи в режимі користувача, інший — у режимі ядра; ці стеки розміщені в адресному просторі процесу, що створив цей потік;
3. локальна пам'ять потоку (TLS);
4. унікальний ідентифікатор потоку (thread id, tid), із того самого простору імен, що й ідентифікатори процесів.

6 Створення потоку

Для створення потоку призначена функція ***CreateThread(),*** а для його завершення — ***EndThread().***

Якщо з коду потоку викликаються функції стандартної бібліотеки мови С, то використовуються бібліотечні функції роботи з потоками з файлу process.h. Це функція ***_beginthreadex()*** для створення потоку та ***_endthreadex()*** — для завершення потоку.

Синтаксис функції *_beginthreadex():*

```
uintptr_t _beginthreadex( // NATIVE CODE
   void *security,
   unsigned stack_size,
   unsigned ( __stdcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);
```

#include <process.h>

*unsigned long _beginthreadex(void *security, unsigned stack_size, unsigned WINAPI (*thread_fun)(void *), void *argument, unsigned init_state, unsigned *tid);*

security — атрибути безпеки потоку; *stack_size* — розмір стека для потоку;

*std_call – (unsigned WINAPI*thread_fun)(void *)*

thread_fun - покажчик на функцію потоку;

arglist — додаткові дані для передачі у функцію потоку;

initflag — початковий стан потоку під час створення;

thrdaddr — покажчик на змінну з ідентифікатором потоку;

Функція *_beginthreadex()* повертає дескриптор створеного потоку, який потрібно перетворити в тип HANDLE:

```
HANDLE th = (HANDLE)_beginthreadex( ... );
```

Дескриптор закривають за допомогою CloseHandle():

```
CloseHandle(th);
```

Приклад виклику _beginthreadex():

```
unsigned tid; int number = 0;
HANDLE th = (HANDLE) _beginthreadex (NULL, 0, thread_fun,
                                     (void *)++number, 0, &tid);
```

Для доступу до дескриптора призначена функція GetCurrentThread():

```
unsigned int WINAPI thread_fun (void *num)
{ HANDLE curth = GetCurrentThread(); }
```

7 Завершення потоків у Win API

Функцію потоку можна завершити двома способами.

1. Виконати у ній оператор return:

```
unsigned WINAPI thread_fun (void *num) { return 0;}
```

2. Викликати функцію _endthreadex() з параметром коду повернення:

```
void _endthreadex(
   unsigned retval
);
```

```
unsigned WINAPI thread_fun (void*num) { _endthreadex(0);}
```

Приклади програм керування процесами та потоками

Приклад 1. Створення потоку функцією CreateThread()

```
#include
<windows.h>
#include
<iostream.h>
#include <conio.h>

volatile int n;

DWORD WINAPI Add(LPVOID iNum)
{
        cout << "Thread is started." <<
        endl;n += (int) iNum;
        cout << "Thread is finished." << endl;
        return 0;
}

int main()
{
        int inc = 10;
        HANDLE
        hThread;
        DWORD
        IDThread;

        cout << "n = " << n << endl;
        //запускаємо потік Add
```

```
        hThread = CreateThread(NULL, 0, Add, (void*)inc, 0, &IDThread);
        if(hThread == NULL)
          return GetLastError();

        //чекаємо, поки потік Add закінчить
        роботуWaitForSingleObject(hThread,
        INFINITE);

        //закриваємо дескриптор потоку
        AddCloseHandle(hThread);
        cout << "n = " << n << endl;

        getch()
        ; return
        0;
}
```

Приклад 2. Зупинка та відновлення потоку функціями SuspendThread(),ResumeThread(). Завершення потоку функцією TerminateThread().

```
#include
<windows.h>
#include
<iostream.h>

volatile UINT nCount;
volatile DWORD
dwCount;void thread()
{
    for(;;)
    {
        nCount++;
        //призупинка потоку на 100
        мілісекундSleep(100);
    }
}

int main()
{
        HANDLE
        hThread;
        DWORD
        IDThread;char c;
```

```
        hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) thread, NULL, 0,
        &IDThread);

if(hThread == NULL) return GetLastError();
        for(;;)
        {
                cout << "Input :" << endl;
                cout << "\t'n' to exit " << endl;
                cout << "\t'y' to display the count " << endl;
                cout << "\t's' to suspend thread " << endl;
                cout << "\t'r' to resume thread " << endl;
                cin >> c;
                if (c=='n')       break;
                switch(c)
                {
                        case 'y':
                          cout << "count =" << nCount << endl;
                          break;
                        case 's':
                          //призупиняємо потік
                          thread dwCount = SuspendThread(hThread);
                          cout << "Thread suspend count =" << dwCount << endl;
                          break;
                        case 'r':
                          //відновляємо потік thread dwCount = ResumeThread(hThread);
                          cout << "Thread suspend count =" << dwCount << endl;
                          break;
                }
        }
                  //перериваємо виконання потоку
                  threadTerminateThread(hThread, 0);

                  //закриваємо дескриптор потоку
                  threadCloseHandle(hThread);
                  return 0;
}
```

Приклад 3. Визначення дескриптора поточного потоку функцією GetCurrentThread().

```
 #include
 <windows.h>
 #include
 <iostream.h>
```

```
int main()
{
        HANDLE hThread;
        hThread = GetCurrentThread();cout << hThread << endl; cin.get();
        return 0;
}
```

Приклад 4. Запуск процесу Notepad

```
#include
<windows.h>
#include
<iostream.h>
#include <conio.h>

int main()
{
 STARTUPINFO si;
 PROCESS_INFORMATION
 pi;

 //заповнюємо значення структури STARTUPINFO за замовчуванням
 ZeroMemory(&si, sizeof(STARTUPINFO));

 si.cb = sizeof(STARTUPINFO);

 //запускаємо процес Notepadif
      (!CreateProcess(
          NULL,
          "Notepad.exe",NULL,
          NULL, FALSE,0, NULL, NULL,
          &si, &pi)
         )
 {
        cout << "The new process is not creared. " << endl
        << "Check a name of the process. " << endl;
        return 0;
 }
 Sleep(1000);
 CloseHandle(pi.hThread);
 CloseHandle(pi.hProcess);

        return 0;
 }
```

Приклад 5. Завершення процесу функцією ExitProcess().

```
#include
<windows.h>
#include
<iostream.h>

volatile UINT
count;void
thread()
{
   for(; ;)
   {
        count++;
        Sleep(100
        );
   }
}
int main()
{
        char c;
        HANDLE hThread;
        DWORD IDThread;

        hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) thread, NULL, 0,
&IDThread);

        if(hThread == NULL)
          return GetLastError();
        for(;;)
        {
             cout << "Input 'y' to display the count or any char to exit: ";
             cin >> (char) c;
             if(c == 'y')
                cout << "count =" << count << endl;
             else
               ExitProcess(1);
          }
}
```

Приклад 6. Визначення дескриптора поточного процесу функцією GetCurrentProcess().

```
#include
<windows.h>
#include
<iostream.h>

int main()
{
        HANDLE hProcess;

        hProcess =

        GetCurrentProcess();cout <<

        hProcess << endl; cin.get();

        return 0;
}
```

Завдання для студентів

1. Розробити програми для роботи з процесами та потоками. Для програмної реалізації використовувати середовище програмування Microsoft Visual Studio, мова програмування – C/C++, інтерфейс консольний.
2. Скласти звіт про виконання лабораторної роботи.

   Зміст звіту:
   - опис функцій для роботи з процесами та потоками;
   - постановка задачі;
   - програмні коди.
   - результати виконання програм;
   - висновок.
3. До захисту лабораторної роботи підготувати відповіді на контрольні питання.

Контрольні питання

1. Назвати складові елементи процесу.
2. Назвати структури даних процесу.
3. Як виконується створення процесів? Дати характеристику функціям, які при цьому використовуються.
4. Як виконується завершення процесів? Охарактеризувати функції, які при цьому використовуються. Проаналізувати параметри функцій.
5. Як отримати дескриптор поточного процесу? Як отримати ідентифікатор поточного процесу? Охарактеризувати функції, які при цьому

використовуються. Проаналізувати параметри функцій.

6. Яка різниця між функціями ExitProcess() та TerminateProcess()?
7. Назвати складові елементи потоку.
8. Як виконується створення потоків? Дати характеристику функціям, які при цьому використовуються. Проаналізувати параметри функцій.
9. Як виконується завершення потоків? Охарактеризувати функції, які при цьомувикористовуються. Проаналізувати параметри функцій.
10. Як виконується зупинка та відновлення потоку? Охарактеризувати функції, які при цьому використовуються. Проаналізувати параметри функцій.

Варіанти завдань:

1. Розробити дві програми. Перша обчислює число Фібоначчі за заданим номером n за формулою $F_n = F_{n-1} + F_{n-2}$, де $F_0 = F_1 = 1$. Обчислення числа Фібоначчі оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
2. Розробити дві програми. Перша шукає найбільше число у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найбільшого оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
3. Розробити дві програми. Перша шукає найбільше число у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найбільшого оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
4. Розробити дві програми. Перша сортує динамічний масив цілих чисел розміру n за зростанням (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Сортування масиву оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
5. Розробити дві програми. Перша обчислює середнє арифметичне елементів динамічного двовимірного масиву цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук середнього арифметичного оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
6. Розробити дві програми. Перша обчислює суму парних чисел від L до U. Обчислення суми оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
7. Розробити дві програми. Перша шукає кількість додатних чисел у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті додатних чисел оформити

як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).

8. Розробити дві програми. Перша шукає кількість повторень заданого числа *x* у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількість повторень заданого числа оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
9. Розробити дві програми. Перша шукає кількість пробілів у текстовому файлі. Пошук кількісті пробілів оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
10. Розробити дві програми. Перша шукає кількість цифр у текстовому файлі. Пошук кількісті цифр оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
11. Розробити дві програми. Перша обчислює суму квадратів елементів динамічного масиву цілих чисел розміру *n (*масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук суми квадратів оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
12. Розробити дві програми. Перша шукає кількість чисел, які більші за своїх сусідів, у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті чисел, які більші за своїх сусідів, оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
13. Розробити дві програми. Перша шукає скільки раз повторюється найбільше число у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількості найбільшого числа оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
14. Розробити дві програми. Перша обчислює кількість дільників заданого числа *n*. Обчислення кількості дільників оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
15. Розробити дві програми. Перша шукає найменше число у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найменшого оформити як функцію потоку.

Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).

16. Розробити дві програми. Перша шукає найменше число у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найменшого оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
17. Розробити дві програми. Перша сортує динамічний масив цілих чисел розміру n за спаданням (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Сортування масиву оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
18. Розробити дві програми. Перша обчислює суму непарних чисел від L до U. Обчислення суми оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
19. Розробити дві програми. Перша шукає кількість від'ємних чисел у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті від'ємних чисел оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
20. Розробити дві програми. Перша шукає кількість нулів у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті нулів оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
21. Розробити дві програми. Перша шукає кількість малих латинських літер у текстовому файлі. Пошук кількісті малих латинських літер оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
22. Розробити дві програми. Перша шукає кількість великих латинських літер у текстовому файлі. Пошук кількісті великих латинських літер оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
23. Розробити дві програми. Перша шукає кількість чисел, які менші за своїх сусідів, у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті чисел, які менші за своїх сусідів оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).

24. Розробити дві програми. Перша шукає порядковий номер найбільшого числа у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук порядкового номеру найбільшого числа оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
25. Розробити дві програми. Перша шукає порядковий номер найменшого числа у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук порядкового номеру найменшого числа оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
26. Розробити дві програми. Перша шукає кількість рядків у текстовому файлі. Пошук кількісті рядків оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
27. Розробити дві програми. Перша обчислює суму чисел у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Обчислення суми чисел як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
28. Розробити дві програми. Перша заміняє усі додатні числа на нуль у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну усіх додатних числа на нуль оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
29. Розробити дві програми. Перша заміняє усі від'ємні числа на нуль у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну усіх від'ємних числа на нуль оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).
30. Розробити дві програми. Перша заміняє усі від'ємні числа на *-1*, а додатні на *1* у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну чисел оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).

Приклад виконання лабораторної роботи:

```
// Програма переписує елементи динамічного масиву розміру n у зворотному порядку
// Переписування елементів масиву у зворотному порядку оформлено у вигляді функції потоку

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

// Структура для передачі аргументів у функцію потоку
struct ThreadArgs {
    int* array;      // Динамічний масив
    int array_size;  // Розмір масиву
};

// Функція потоку для обертання масиву
DWORD WINAPI reverseArray(LPVOID lpParam) {
    struct ThreadArgs* threadArgs = (struct ThreadArgs*)lpParam;
    int* array = threadArgs->array;
    int size = threadArgs->array_size;

    for (int i = 0; i < size / 2; i++) {
        // Обмін елементів для обертання масиву
        int temp = array[i];
        array[i] = array[size - i - 1];
        array[size - i - 1] = temp;
    }

    return 0;
}

int main() {
    int n;
    printf("Enter the size of the array: ");
    scanf_s("%d", &n);

    // Генерування динамічного масиву з випадковими значеннями
    int* dynamicArray = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        dynamicArray[i] = rand() % 100;  // Випадкові значення від 0 до 99
    }

    // Виведення початкового масиву
    printf("\nOriginal array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", dynamicArray[i]);
    }
    printf("\n");

    // Створення структури з аргументами для функції потоку
    struct ThreadArgs threadArgs;
    threadArgs.array = dynamicArray;
    threadArgs.array_size = n;

    HANDLE hThread;
    DWORD dwThreadId;

    // Створення потоку для обертання масиву
    hThread = CreateThread(
        NULL,              // default security attributes
        0,                 // use default stack size
        reverseArray,      // thread function
        &threadArgs,       // argument to thread function
```

```
        0,                  // use default creation flags
        &dwThreadId);       // returns the thread identifier

    if (hThread == NULL) {
        fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());
        return 1;
    }

    // Чекаємо завершення потоку
    WaitForSingleObject(hThread, INFINITE);

    // Виведення обернутого масиву
    printf("\nReversed array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", dynamicArray[i]);
    }
    printf("\n");

    // Виведення інформації про потік та основний процес
    printf("\nThread ID: %lu\n", dwThreadId);
    printf("Thread Handle: %p\n\n", hThread);
    printf("Main process ID: %lu\n", GetCurrentProcessId());
    printf("Main thread ID: %lu\n", GetCurrentThreadId());
    printf("Main process Handle: %p\n", GetCurrentProcess());
    printf("Main thread Handle: %p\n", GetCurrentThread());

    // Закриття дескрипторів, які більше не потрібні
    CloseHandle(hThread);
    free(dynamicArray);

    getchar(); getchar();

    return 0;
}


// Програма запускає файл Lab1.exe як новостворений процес
#include <stdio.h>
#include <windows.h>

int main() {
    LPCWSTR lpApplicationName = L"C:\\source\\repos\\Lab1\\Debug\\Lab1.exe";
    PROCESS_INFORMATION pi;
    STARTUPINFO si;
    int status = 0;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Встановлюємо прапорець CREATE_NEW_CONSOLE, щоб створити нове консольне вікно
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_SHOWDEFAULT;

    if (!CreateProcess(
        lpApplicationName,
        NULL,
        NULL,
        NULL,
        FALSE,
        CREATE_NEW_CONSOLE,  // Важливо встановити цей прапорець для створення нового
консольного вікна
        NULL,
```

```
    NULL,
    &si,
    &pi)) {
    fprintf(stderr, "CreateProcess failed (%lu).\n", GetLastError());
    return 1;
}

printf("Child process started...\n");

// Виведення інформації про новий процес та батьківський процес
printf("\nChild process ID: %lu\n", pi.dwProcessId);
printf("Child thread ID: %lu\n", pi.dwThreadId);
printf("Child process Handle: %p\n", pi.hProcess);
printf("Child thread Handle: %p\n", pi.hThread);

// Виведення інформації про батьківський процес
printf("\nParent process ID: %lu\n", GetCurrentProcessId());
printf("Parent thread ID: %lu\n", GetCurrentThreadId());
printf("Parent process Handle: %p\n", GetCurrentProcess());
printf("Parent thread Handle: %p\n", GetCurrentThread());


// Очікування завершення дочірнього процесу
WaitForSingleObject(pi.hProcess, INFINITE);

// Отримання статусу завершення процесу
GetExitCodeProcess(pi.hProcess, (LPDWORD)&status);
printf("\nChild process finished with status %d\n", status);

// Закриття дескрипторів, які більше не потрібні
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);

return 0;
}
```