Практична робота №7

Створення і видалення процесу в ОС Windows

Мета: Навчитись створювати і видаляти процеси в ОС Windows, використовуючи Assembler для Win 32.

Завдання.

Створити новий процес: розробити програму мовою Assembler для Win 32, яка створює процес, перериває процес, знищує процес.

Теоретичні відомості

Створення і керування процесом в ОС Windows.

Процес – це додаток, що виконується, і який складається із особового віртуального адресного простору, коду даних та інших ресурсів операційної системи, таких як файли, синхронізаційні об'єкти, що відомі для процесу.

Процес має складатися із використовуваного модуля, особистого адресного простору і гілки. У кожного процесу має бути хоча б одна гілка. Фактично гілка – це черга, що виконується.

Коли ОС Windows вперше створює процес, вона створює тільки одну гілку на процес, ця гілка зазвичай починає виконання з першої інструкції в модулі, якщо в подальшому знадобиться більше гілок він може сам створювати його. Коли Windows отримає команду для створення процесу, то вона створює особистий адресний простір для процесу, а потім завантажує виконуваний файл в просторі. Після цього ОС створює основну гілку для процесу.

Далі представлено синтаксис функції створення процесу в ОС Windows CreateProcess:

```
   CreateProcess proto lpAplicationName: DWORD, \
lpComandLine:DWORD, \
   lpProcessAttributes:DWORD, \ lpThreatAttributes:DWORD, \
   bInheritHandles: DWORD, \ dwCreationFlags: DWORD, \
lpEnviroment: DWORD,\
   lpCurrentDirectory: DWORD, \ lpStartupInfo: DWORD, \
lpProcessInformation: DWORD
```

Більшу частину параметрів можна ігнорувати.

lpAplicationName – Ім'я виконуючого файлу з або без шляху. Якщо параметр рівний нулю, маємо представити ім'я виконуючого файлу в параметрі lpComandLine.

lpComandLine – Аргументи командного рядка до програми, яку необхідно запустити.

lpProcessAttributes і *lpThreatAttributes* – Тут необхідно вказати атрибути безпеки для процесу і головної гілки. Якщо вони дорівнюють нулям (NULL), то використовуємо атрибути безпеки за замовчуванням.

bInheritHandles – прапор, який вказує, чи буде новий процес наслідувати всі відкриті *Handle* вашого процесу.

dwCreationFlags – декілька прапорів, які визначають поведінку процесу при його створенні, наприклад процес може бути створений і тут же призупинений, щоб була змога перевірити його або змінити перш ніж він зупиниться.

lpEnviroment – Вказівник на блок пам'яті, який містить декілька змінних оточень для нового процесу.

lpCurrentDirectory – Вказівник на рядок, який вказує на поточний диск і директорія для дочірнього процесу. NULL, якщо дочірній процес має унаслідувати її від батьківського процесу.

lpStartupInfo – Вказує на структуру *STARTUPINFO*, яка визначає як має з'явитися основне вікно нового процесу.

lpProcessInformation – вказує на структуру *PROCESS_INFORMATION*, яка отримує ідентифікаційну інформацію про новий процес.

Структура *PROCESS_INFORMATION* має наступні параметри.

```
PROCESS_INFORMATION STRUCT
hProcess     HANDLE ?;        хендл дочірнього процесу
process
   hThread    HANDLE ?;       хендл основної гілки дочірнього процесу
dwProcessId     DWORD ?;   ID дочірнього процесу
dwThreadId      DWORD ?;     ID основної гілки
PROCESS_INFORMATION ENDS
```

Хендл процесу і ID процесу – це різні речі. **ID процесу** – це унікальний ідентифікатор в системі. **Хендл процесу –** це значення, що повертає Windows для

використання іншими API-функціями, зв’язаними з процесами. Хендл процесу не може використовуватись для ідентифікації процесу, так як він не унікальний.

Після виклику функції *CreateProcess*, створюється новий процес і функція відразу повертається. Можна перевірити, чи є даний процес активним, викликавши функцію *GetExitCodeProcess,* яка має наступний синтаксис:

GetExitCodeProcess proto hProcess:DWORD, lpExitCode:DWORD

Якщо виклик цієї функції успішний, *lpExitCode* міститиме код виходу із процесу, що запитується. Якщо значення *lpExitCod*e дорівнює STILL_ACTIVE, тоді це означає, що процес все ще запущений. Можна примусово перервати процес, викликавши функцію TerminateProcess.

Хід виконання роботи.

Створюємо в блокноті файл MSGBOX.ASM:

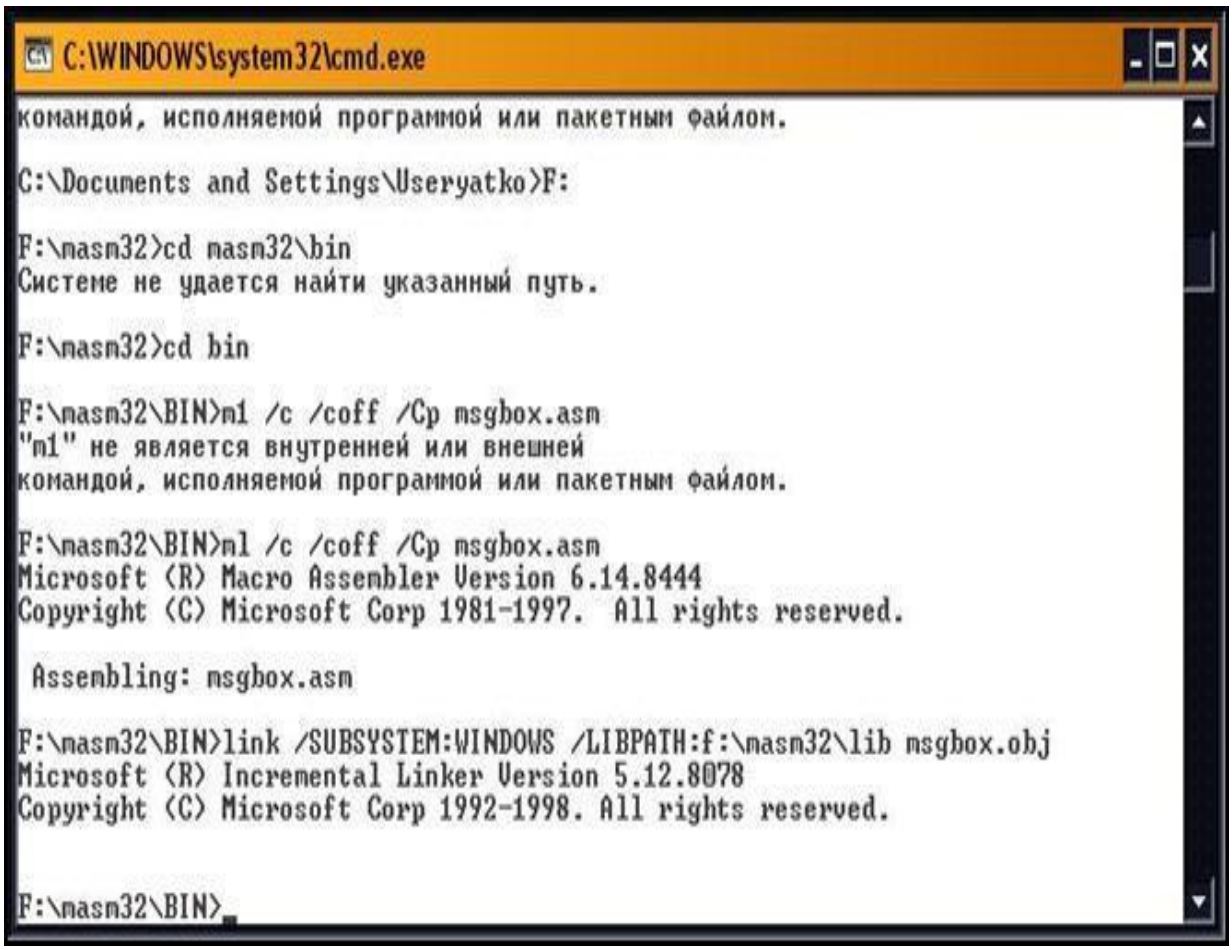
```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
MsgCaption db "Нове вікно",0
MsgBoxText db "Створено новий процес. Програма MSGBOX.EXE запущена на виконання і працює.",0

.code
start:
    invoke MessageBox, NULL, addr MsgBoxText, addr MsgCaption,
        MB_OK invoke ExitProcess, NULL
end start
```

Компілюємо його щоб отримати виконуваний файл MSGBOX.EXE

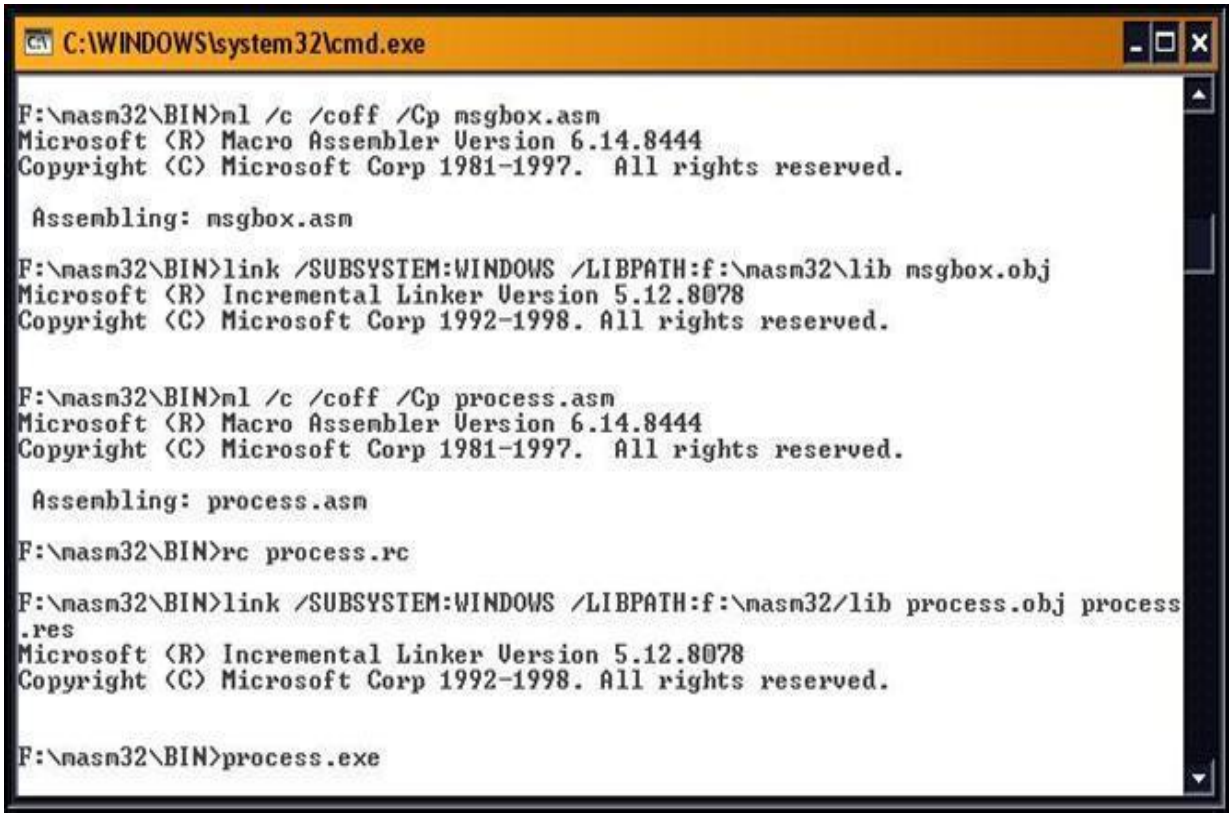


Створюємо у блокноті файл ресурсів PROCESS.RC

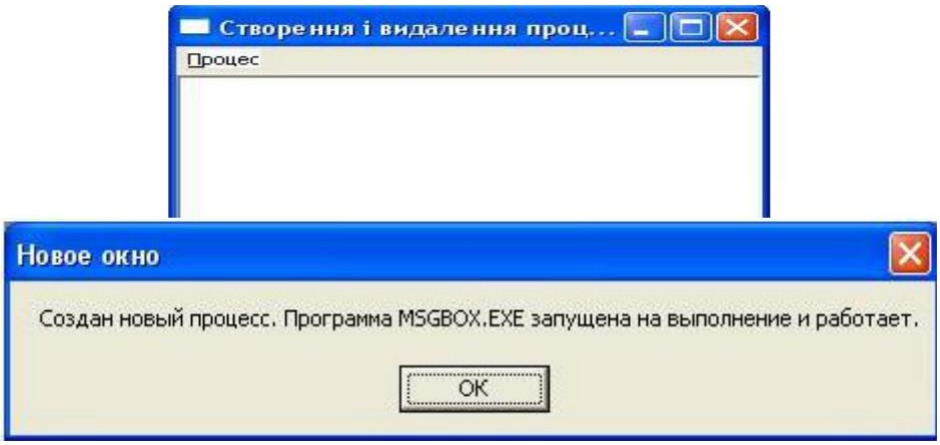
```
#define IDM_CREATE_PROCESS 1
#define IDM_TERMINATE 2
#define IDM_EXIT 3

FirstMenu MENU
{
POPUP "&Процес"
        {
        MENUITEM "&Створити процес",IDM_CREATE_PROCESS
        MENUITEM "&Перервати процес",IDM_TERMINATE
                MENUITEM SEPARATOR
                MENUITEM "&Вихід",IDM_EXIT
        }
}
```

Створюємо у блокноті файл основної програми PROCESS.ASM.
Компілюємо його:



Запускаємо на виконання PROCESS.EXE і отримуємо результати:



Підсвітивши заголовок вікна створення і видалення процесу, отримують доступ до пункту “Перервати процес” спливаючого вікна. Натискання на “Перервати процес” призводить до закривання “Нове вікно”.

Контрольні питання

Що таке процес в ОС?

Що включають в себе атрибути процесу?

Які існують переходи між станами процесів?

Що містить образ процесу?

Які дії виконує ОС при створенні нового процесу?

Які існують підходи до розробки ОС?

```
    CreateThread proto LpThreadAttributes:DWORD,\
    dwStackSize: DWORD,\ LpStartAddress:
DWORD,\LpParameter: DWORD,\dwCreationFlags: DWORD,\
LpThreadid: DWORD
```

Функція *CreateThread* схожа на *CreateProcess*.

lpThreadAttributes - можливо використовувати NULL, якщо необхідно, щоб у потоку були установки безпеки за замовчуванням.

dwStackSize – вказати розмір стека потоку. Якщо необхідно, щоб потік мав такий же розмір стека, як і у основного, використовують NULL в якості параметру.

lpStartAddress – адреса функції потоку. Ця функція буде виконувати призначену для потоку роботу. Ця функція має отримувати один і тільки один 32-бітний параметр і повертати 32-бітне значення.

lpParametr – параметр, який необхідно передати функції потоку.

dwCreationFlags – 0 означає, що потік починає виконуватися відразу після створення. Для зворотнього можна використовувати прапор *CREATE_SUSPEND*.

lpThreadId – CreateThread – помістити сюди *ID* створеного потоку.

Якщо виклик *CreateThread* пройшов успішно, вона повертає хэндл створеного потоку, в зворотньому випадку вона повертає *NULL*.

Функція потоку запускається так швидко, як тільки закінчується виклик *CreateThread*, якщо тільки не позначено прапор *CREATE_SUSPEND*. В цьому випадку потік буде заморожений до виклику функції *ResumThread*.

Коли функція потоку повертається (за допомогою інструкції *ret*) Windows опосередковано викликає *ExitThreat* для функції потоку.

Можливо отримати код виходу потоку за допомогою функції *GetExitCodeThread*.

Існує три методи комунікації між потоками:

Використання глобальних змінних.

Windows повідомлення.

Події.

Потоки розглядають ресурси процесу, включаючи глобальні змінні, тому потоки можуть використовувати їх для того, щоб взаємодіяти один з одним. Але все таки, цей метод має використовуватись обережно. Синхронізацію потрібно уважно спланувати.

Також можливо використовувати Windows повідомлення, щоб виконувати взаємодію між потоками.

Якщо всі потоки мають користувацький інтерфейс, то немає проблем: цей метод може використовуватися для двохсторонньої комунікації. Все що потрібно зробити – це визначити один або більше додаткових Windows повідомлень, які будуть використовуватися з потоками. Визначаємо повідомлення використовуючи значення

WM_USER як базове, наприклад так:

WM_MYCUSTOMMSG equ WM_USER+100h

Якщо один із потоків має інтерфейс користувача, а інший є робочим ми не можемо використовувати даний метод для двостороннього користування, так як робочого потоку немає свого вікна, а відповідно і черги повідомлень. Можливо використовувати наступні схеми:

Потік з інтерфейсом користувача → глобальна змінна → робочий потік.

Робочий потік → Windows –повідомлення → Потік з користувацьким інтерфейсом.

Фактично ми будемо використовувати цей метод в нашому прикладі.

Хід виконання роботи

Створюємо в блокноті файл potok.asm.

```
386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.const
IDM_CREATE_THREAD equ 1
IDM_EXIT equ 2
WM_FINISH equ WM_USER+100h
```

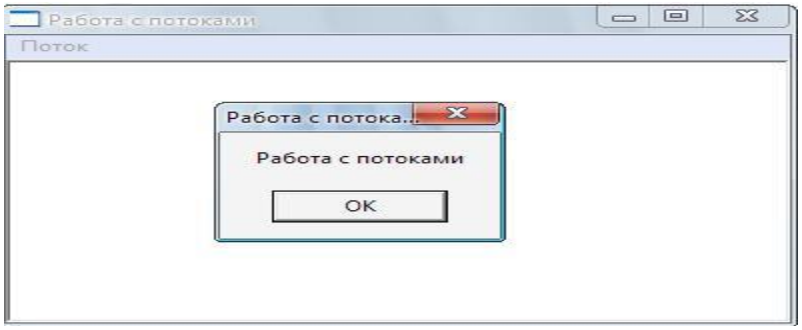
```
.data
……
……
……


ThreadProc PROC USES ecx
        Param:DWORD mov ecx,600000000
Loop1:
        add eax,eax
        dec ecx
        jz Get_out
        jmp Loop1
Get_out:
        invoke
        SendMessage,hwnd,WM_FINISH,NULL,NULL ret
ThreadProc ENDP
end start
```

Створюємо в блокноті файл potok.rc.

```
#define IDM_CREATE_THREAD 1
#define IDM_EXIT 2
FirstMenu MENU
{
POPUP "&Поток"
        {
        MENUITEM "&Створити поток",IDM_CREATE_THREAD
        MENUITEM SEPARATOR
       MENUITEM "&Вихід",IDM_EXIT
        }
}
```

Результат компіляції і запуску програми.



Контрольні питання

Що таке потік?

Назвіть елементи, що використовуються потоками одночасно?

Назвіть елементи, що використовуються потоками нарізно?

Які існують способи реалізації потоків?

Які переваги реалізації потоків в просторі користувача?

В яких ОС можлива реалізація паралельного виконання команд різних потоків?