Міністерство освіти і науки України
Кіровоградський національний технічний університет

Методичні вказівки

до лабораторних робіт
з предмету «Системне програмування» (6 семестр)
для студентів денної та заочної форми навчання
напряму 050102 «Комп'ютерна інженерія»

Розробили:
доцент Коваленко О.В.
викладач Константинова Л.В.

Кіровоград 2016

Методичні вказівки до лабораторних робіт з предмету «Системне програмування» (6 семестр) для студентів денної та заочної форми навчання напряму 050102 «Комп'ютерна інженерія». Укл. Коваленко О.В., Константинова Л.В. - Кіровоград: КНТУ, 2016.- 43 с.

Укладачі: Коваленко О.В., Константинова Л.В.

Рецензент:_____

Рекомендовано кафедрою програмування та захисту інформації, протокол № __ від ___ _____ 2016 р.

Вступ

Знання основних принципів системного програмування необхідно при розробці програм для роботи з апаратним забезпеченням (таких, як плати ЦАП/АЦП, інтерфейси вводу-виводу тощо) та є надзвичайно корисним і при розробці звичайних прикладних програм, які не здійснюють прямої взаємодії з апаратним забезпеченням. Вивчення внутрішньої архітектури програми дозволяє не лише краще розуміти спосіб її функціонування, але й використовувати в її роботі різноманітні нестандартні засоби, які при застосовуванні середовищ візуального програмування (таких як Delphi, C++ Builder тощо) найчастіше приховані від розробника.

В методичних вказівках представлено теоретичний і практичний матеріал для виконання завдань з лабораторних робіт з системного програмування. Дані методичні вказівки продовжують методичні вказівки до лабораторних робіт з предмету «Системне програмування (5 семестр) і розглядають таку тематику теоретичних та практичних питань як, процеси та потоки, базова структура прикладної програми, віртуальний адресний простір, взаємодія між процесами, служби. Приводиться багато практичних прикладів та ілюстрацій, які допомагають краще засвоїти викладений матеріал.

Теми лабораторних робіт, що розглядаються та оцінювання знань

Пропонується виконати лабораторні роботи за наступними темами:

Теми лабораторних робіт	Години
1. Процеси, потоки, базова структура ПЗ.	2
2. Механізм керування пам'яттю.	2
3. Взаємодія між процесами IPC (Частина 1).	2
4. Взаємодія між процесами IPC (Частина 2).	2
5. Взаємодія між процесами IPC (Частина 3).	2
6. Системна взаємодія між процесами.	2
7. Основи використання служб ОС WINDOWS	2
Всього:	14

Форма підсумкового контролю: залік (п’ятий семестр), іспит (шостий семестр).

Максимальну кількість балів студент може одержати у випадку відвідування всіх лекцій, лабораторних занять, виконання і захисту самостійної роботи у встановлений термін, проходження контролю.

При виконанні і при захисті лабораторних робіт після встановленого терміну, одержані бали перераховуються з коефіцієнтом: для самостійної роботи студента -0,3; лабораторної роботи -0,7.

В якості самостійного завдання необхідно виконати теоретичну роботу згідно обраної студентом теми.

Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою
		для екзамену, курсової роботи
90 – 100	**A**	відмінно
82-89	**B**	добре
74-81	**C**	
64-73	**D**	задовільно
60-63	**E**	
35-59	**FX**	незадовільно з можливістю повторного складання
0-34	**F**	незадовільно з обов’язковим повторним вивченням дисципліни

Вибравши зручне для Вас середовище швидкої розробки додатків, узгодивши його з лектором, ви повинні виконати завдання до лабораторних робіт, а також відповісти на питання вкінці кожної лабораторної роботи. Звіт повинен містити хід виконання завдань, лістинг програми а також графічні матеріали, що підтверджують виконання цих завдань.

Лабораторна робота №1 (семестр 2)

ТЕМА: ПРОЦЕСИ, ПОТОКИ, БАЗОВА СТРУКТУРА ПЗ

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

Процеси та потоки

Процес забезпечує ресурси, необхідні для виконання певної програми. Процес має власний віртуальний адресний простір, виконавчий код, дані, ідентифікатори об’єктів, змінні оточення та пріоритет. Кожен процес стартує з єдиним потоком, який ще називають первинним, але може створювати додаткові потоки. Всі потоки процесу розподіляють його віртуальний адресний простір і системні ресурси. Кожен потік має ідентифікатори виняткових ситуацій, пріоритет і множину структур, які система використовує для збереження контексту потоку, поки він призупиняється. У контекст потоку входить множина машинних регістрів, стек ядра, блок оточення потоку і стек користувача в адресному просторі процесу, якому належить даний потік.

Пріоритети процесів та потоків

Кожен потік має певний пріоритет. Діапазон рівнів пріоритету – від 0 (найменший пріоритет) до 31 (найвищий пріоритет). Пріоритет кожного потоку визначається такими критеріями: класом пріоритету його процесу та рівнем пріоритету потоку в межах класу пріоритету процесу. Клас пріоритету та рівень пріоритету утворюють базовий пріоритет потоку.

Кожен процес має один із таких класів пріоритету:

IDLE_PRIORITY_CLASS,

BELOW_NORMAL_PRIORITY_CLASS,

NORMAL_PRIORITY_CLASS,

ABOVE_ NORMAL_PRIORITY_CLASS,

HIGH_ PRIORITY_CLASS,

REALTIME_ PRIORITY_CLASS.

За замовчуванням клас пріоритету набуває значення NORMAL_PRIORITY_CLASS. При створенні процесу функцією *CreateProcess* можна задавати клас пріоритету. Для зміни класу пріоритету існуючого процесу використовується функція *SetPriorityClass*, для визначення пріоритету існуючого процесу – функція *GetPriorityClass*.

Процеси, якими керує система, можуть використовувати найнижчий IDLE_PRIORITY_CLASS, що запобігатиме конфлікту таких процесів із процесами вищого пріоритету.

Якщо виконується потік процесу з класом HIGH_ PRIORITY_CLASS, інші потоки в системі не отримуватимуть процесорного часу. Якщо кілька потоків одночасно мають такий пріоритет, вони втрачають свою ефективність. Клас найвищого пріоритету використовується для потоків, які повинні відповідати на найважливіші події. Якщо прикладна програма виконує завдання, яке потребує цього класу пріоритету, слід використовувати функцію *SetPriorityClass* для тимчасового підвищення класу пріоритету і відразу після виконання

завдання відновлювати звичайний пріоритет. Інший спосіб для вирішення подібного завдання – створити процес високого пріоритету, всі потоки якого будуть блокованими більшість часу і тимчасово розблоковуватимуться при виникненні певної події.

Клас пріоритету REALTIME_ PRIORITY_CLASS припиняє виконання системних потоків, які керують вводом від миші, клавіатури та фоновим записом на диск, що може призвести до втрати даних або не функціональності всієї системи. Цей клас може використовуватися для програм, які повинні працювати безпосередньо з апаратним забезпеченням або які виконують швидкі, обмеженні в часі завдання.

Функція CreateProcess

Створення процесу здійснюється викликом однієї з таких функцій, як *CreateProcess*, *CreateProcessAsUser*, *CreateProcessWithTokenW* або *CreateProcessWithLofon*, і проходить у кілька етапів за участю трьох компонентів операційної системи: бібліотеки клієнтської частини Windows (Kernel32.dll), виконавчої системи й процесу підсистеми оточення Windows (Csrss). Оскільки архітектура Windows підтримує кілька підсистем оточення, операції, необхідні для створення об’єкта «процес» виконавчої системи (яким можуть користуватися й інші підсистеми оточення), відділені від операцій, необхідних для створення процесу. Тому частина дій функцій *CreateProcess* специфічна для семантики, що вноситься підсистемою Windows.

У наведеному нижче списку перераховані основні етапи створення процесу функцією API *CreateProcess*:

1. Відкривається файл образу (EXE), що буде виконувати в процесі.
2. Створюється об’єкт «процес» виконавчої системи.
3. Створюється первинний потік (стек, контекст і об’єкт «потік» виконавчої системи).

4. Підсистема Windows повідомляється про створення нового процесу й потоку.
5. Починається виконання первинного потоку (якщо не зазначено прапорець CREATE_SUSPENDED).
6. У контексті нового процесу й потоку ініціалізується адресний простір (наприклад, завантажуються необхідні DLL) і починається виконання програми.

При виклику *CreateProcess* клас пріоритету вказується в параметрі CreationFlag, при цьому можна задати відразу кілька класів пріоритету. Windows вибирає найнижчий з них.

Коли для нового процесу не вказується клас пріоритету, за замовчуванням приймається Normal, якщо клас пріоритету батьківського процесу не дорівнює Idle або Below Normal. В останньому випадку новий процес отримує той же клас пріоритету, що й у батьківського процесу.

Якщо для нового процесу зазначено клас пріоритету Realtime, а творець не має привілеїв Increase Scheduling Priority, встановлюється клас пріоритету High. Інакше кажучи, функція *CreateProcess* завершується успішно, навіть якщо в того, хто її викликав, недостатньо привілеїв для створення процесів із класом пріоритету Realtime, - просто клас пріоритету нового процесу буде нижчий.

Усі створені процесом вікна пов'язуються з об'єктами "робочий стіл", які є графічним поданням робочого простору. Якщо при виклику *CreateProcess* не зазначено конкретний об'єкт "робочий стіл", новий процес зіставляється з поточним об'єктом "робочий стіл" батьківського процесу.

Крок 1: відкриття образу для виконання

Спочатку *CreateProcess* повинна знайти потрібний виконавчий образ, який буде виконуватись. У Windows XP і новіших ця функція перевіряє, чи не забороняє політика безпеки на даній машині запуск цього образу.

Крок 2: створення об'єкта "процес"

До початку другого етапу функція *CreateProcess* вже відкрила припустимий виконавчий файл Windows і створила об'єкт "розділ" для його проектування на адресний простір нового процесу. Після цього вона створює об'єкт "процес", щоб запустити образ викликом внутрішньої функції *NtCreateProcess.*

Крок 3: створення первинного потоку, його стеку й контексту

До початку третього етапу об'єкт "процес" виконавчої системи повністю ініціалізований. Але у нього ще немає жодного потоку, тому він не може нічого виконувати. Перед тим як створити потік, потрібно створити стек і контекст, у якому він буде виконуватися.

Далі створюється первинний потік викликом *NtCreateThread*. Потік створюється в призупиненому стані й відновлюється лише після завершення ініціалізації процесу.

Крок 4: повідомлення підсистеми Windows про новий процес

Якщо задані відповідні правила, для нового процесу створюється маркер з обмеженими правами доступу. До цього моменту всі необхідні об'єкти виконавчої системи створені, і Kernel32.dll посилає підсистемі Windows повідомлення, щоб вона підготувалася до виконання нового процесу й потоку. Повідомлення включає таку інформацію:

- дескриптори процесу й потоку;
- прапорці створення;
- ідентифікатор батьківського процесу;
- прапорець належності даного процесу до Windows-програм.
-

Крок 5: запуск первинного потоку

До початку цього етапу оточення процесу вже створено, його потоку виділені ресурси, а підсистемі Windows відомий факт існування нового процесу. Тому для завершення ініціалізації нового процесу відновлюється

виконання його первинного потоку, якщо тільки при його створенні не зазначено прапорець CREATE_SUSPENDED.

Крок 6: ініціалізація в контексті нового процесу

Новий потік починається з виконання стартової процедури потоку режиму ядра. А коли процедура ініціалізації повертає керування у користувацький режим, починається виконання образу в користувацькому режимі і викликається функція потоку.

Приведено простий програмний код для створення нового процесу (запуску програми, виконавчий файл якої називається APINewProcess.exe).

Лістинг 3.1. Створення процесу функцією *CreateProcess*

```
var SI: TStartupInfo;
    PI: TProcessInformation;
.....................................................
    FillCgar(SI,Sizeof(SI),#0);
     SI.cb := Sizeof(SI);
     SI.dwX := 50;
     SI.dwY := 50;
     SI.dwFlags := STARTF_USEPOSITION;
     if not CreateProcess('APINewProcess.exe',nil,nil,nil,false,
           Normal_PRIORITY_Class,nil,nil,SI,PI) then
       MessageBox(Wnd, 'Не вийшло створити процес.','',
        NB_OK or MB_ICONERROR);

 ....Після закінчення роботи з процесом.......

 CloseHandle(PI.hThread);
 CloseHandle(PI.hProcess);
```

Відкриття дескриптора процесу

Якщо необхідно працювати із зовнішнім процесом, потрібно спочатку відкрити дескриптор об'єкта процесу за допомогою функції *OpenProcess:*

```
function OpenProcess(
dMDesiredAccess:  DWORD;  //  Параметри  доступу  для  відкриття  процесу
bInheritHandle: BOOL;   // Параметр успадкування дескриптора
dwProceseId: DWORD        // Ідентифікатор процесу
): THandle;
```

Ця функція за відомим ідентифікатором процесу повертає його дескриптор, який може використовуватися для маніпуляцій над процесом.

Отримання списку запущених процесів

Список запущених процесів можна отримати за допомогою функцій бібліотеки Process Status Helper (PSAPI), яка містить множину функцій для отримання інформації про процеси та драйвери пристроїв.

Для перелічування процесів бібліотека забезпечує функцію E*numProcesses*, яка повертає масив ідентифікаторів процесів.

Функція *EnumProcesses* не дозволяє довідатися, скільки місця потрібно для того, щоб прийняти весь масив ідентифікаторів. Тому слід або програмно задавати такий розмір масиву, який гарантовано перекриє кількість усіх процесів, або викликати функцію в циклі, збільшуючи обсяг буфера, поки розмір повернутого масиву не буде меншим від розміру буфера.

Лістинг 3.3. Отримання списку працюючих процесів

```
var
      PrIDs: array[0...$FFF] of dword; // Масив для зберігання
                                       // ідентифікаторів процесів
      Cnt, ModCnt: cardinal;
      ProcCnt, I: word;
      hProceas, hMod: THandle;
      ModuleName: array[0..MAX_PATH] of char;
      begin
            lbProcList.Items.Clear;

            EnumProcesses(@PrIDs, SizeOf (PrIDs), Cnt);

            ProcCnt := Cnt div SizeOf(DWORD); // Кількість записів про
                                              //процеси
            for I :=0 to ProcCnt - 1 do begin
              case PrIDs[I] of
                0 : IbProcList.Items.Add('Процес Idle (недіяння
                    системи)');
                4 : IbProcList.Items.Add('Процес System');
                else begin
                  hProcess := OpenProcess(
```

```
          PROCESS_QUERY_INFORMATION or PROCESS_VM_READ, False,
                PrIDs[I]);
          if hProcess <> 0 then begin
            EnumProcessModules(hProcess, @hMod, SizeOf(hMod),
                ModCnt);
           GetModuleFilenameEx(hProcess, hMod, ModuleName,
             SizeOf(ModuleName));
           IbProcList.Items.Add(ModuleName);
           CloseHandle(hProcess);
       end    else    IbProcList.Items.Add('-=Назву    отримати
  неможливо=-');
   end;
  end;
end;
```

Потоки у Windows

У Windows реалізовано підсистему витісняючого планування на основі рівнів пріоритету, у якій завжди виконується готовий до виконання потік з найбільшим пріоритетом.

Рівні пріоритету потоків

Як показано на рис. 1, у Windows передбачено 32 рівні пріоритету - від 0 до 31. Ці значення групуються так:

- шістнадцять рівнів реального часу (16-31);
- п'ятнадцять динамічних рівнів (1-15);
- один системний рівень (0), зарезервований для потоку обнуления сторінок пам'яті (zero page thread).

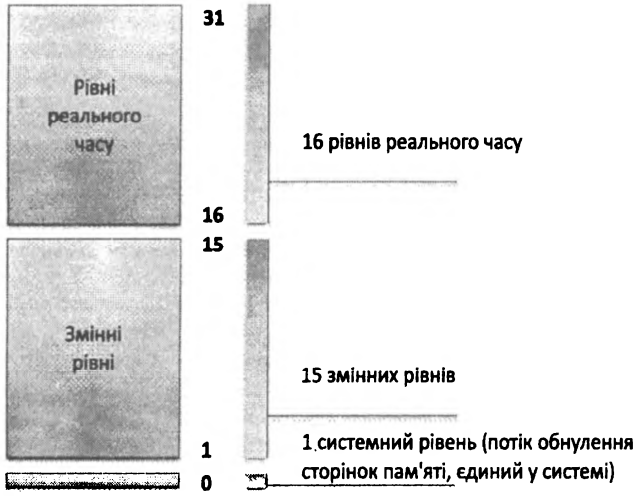

Рис. 1 Рівні пріоритету потоків

Доцільне використання потоків може значно поліпшити продуктивність і зручність використання програм.

Створення потоку

Життєвий цикл потоку починається при його створенні програмою. Запит на створення потоку надходить виконавчій системі Windows, диспетчер процесів виділяє пам'ять для об'єкта "потік" і викликає ядро для ініціалізації

блоку потоку. Нижче перераховані основні етапи створення потоку функцією *CreateThread* (вона знаходиться в Kernel32.dll):

1. Створення стеку користувацького режиму в адресному просторі процесу.

2. Ініціалізація апаратного контексту потоку, специфічного для конкретної архітектури процесора.

3. Для створення об'єкта "потік" виконавчої системи викликається *NtCreateThread. CreateThread* повідомляє підсистему Windows про створення нового потоку і та виконує підготовчі операції.

4. Викликаючому коду повертаються дескриптор і ідентифікатор потоку, згенерований на попередньому етапі.

5. Виконання потоку відновлюється і йому може бути виділено процесорний час (якщо він не був створений із прапорцем CREATESUSPENDED).

Хоча Win32 API забезпечує вичерпну підтримку багатопотоковості, для створення й знищення потоків у VCL є зручний клас - TThread, який надає високорівневий підхід, значно спрощує роботу при використанні у потоках VCL й допомагає уникнути деяких складних моментів. При роботі у Delphi в більшості випадків рекомендується використовувати саме його. Для створення класу, який представлятиме користувацький потік, слід вибрати пункт меню File/ New, а потім Thread Object.

Приклад. Використання класу TThread для створення потоку

Модуль головної форми

```
unit Main;

interface

uses ... TheThread ... ;

const WM_ThreadDone = WM_USER;

type
  TfMain = classes(Tform)
    btnStartThread: TButton;
    st: TStaticText;
    procedure btnStartThreadClick(Sender: TObject);
  private
    T2: TMyThread;
    procedure ThreadDone(var AMessage: TMessage);
message WM_ThreadDone;
  end;
var fMain: TfMain;

impleaentation

{$R * .DFM)

procedure TfMain.btnStartThreadCliclk(Sender: TObject);
begin
  btnStartThread.Enabled:=false;
```

```
  T2 := TMyThread.Create(False);
end;
procedure TfMain.ThresdDone(var AMessage : TMessage);
begin
  btnStartThread.Enabled:=true;
end;
end.
```

Модуль потоку

```
unit TheThread;

interface

uses Classes;

type
  TMyThread = class (TThread)
  private
      FCount: Integer;
      procedure UpdateData;
  protected
      procedure Execute; override;
  end;

implementation

uses Main, SysUtils, Windows;

procedure TMyThread.UpdateData;
begin
  fMain.st.Caption := IntToStr(FCount);
end;

procedure TMyThread.Execute;
var i: Integer;
begin
  Windows.InvalidateRect(fMain.Handle, nil, True);
  for i 1 to 2000000 do
    if i mod 10 = 0 then begin
       FCount := i;
       Synchronise(UpdateData);
    end;
   PostMessage(fMain.Handle, WM_ThreadDone, 0, 0);
end;
end.
```

Синхронізувати головний потік VCL з додатковим слід за допомогою методу *Synchronize* (лістинг 3.11). Проте це не єдиний спосіб: для синхронізації можна використати особливість надсилання повідомлення вікну за допомогою функції *SendMessage*.

Приклад. Узгодження роботи потоків за допомогою функції SendMessage

```
   unit Main;

   interface

uses Windows, Massages, SysUtils, Classes, Graphics,
     Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
   type
     TfMain = class(TForm)
       btnStartThread: TButton;
       st: TStaticText;
       procedure btnStartThreadClick(Sender: TObject);
     private
   end;

   var fMain: TfMain;

   implementation

   {$R *.DFM}

   function ThreadTunc(Param: Integer) :Integer;
   var
     i : integer;
     s : string;
   begin
     for i:= 0 to 2000000 do
      if mod 10 = 0 then begin
     s := IntToStr(i);
     SendMessage(Param, WM__SETTEXT, 0, Integer(PChar(s)));
end;
    Result := 0;
end;

   pocedure TfMain.btnStartThreadClick(Sender: TObject);
   var
     h: THandle;
```

```
  Threadld :Cardinal;
begin
  h:= BeginThread (nil, 128, @ThreadFunc, Pointer(st.Handle),
      0, Threadld);
  CloseHandle(h)
end;

end.
```

Базова структура прикладної програми

У середовищі Windowsможуть працювати прикладні програми різних видів: стандартні віконні програми, консольні програми, служби тощо. Розглянемо структуру віконної програми, основаної на графічному інтерфейсі користувача.

Вікна

Вікно(window) - спеціальний програмний об'єкт, призначений для організації взаємодії програми з операційною системою

Кожне вікно грає певну роль у відображенні результатів на екрані та отриманні інформації від користувача.

Windowsмістить стек вікон, який є множиною всіх створених у системі вікон. Вікна стеку орієнтуються вздовж уявної осі Z, перпендикулярної до площини екрана. Положення вікна у стеку характеризується Z-порядком. Вікно, яке має вищий Z-порядок, перекриває інші вікна.

Крім головного вікна програми використовують кілька інших типів вікон: елементи керування, діалогові вікна та вікна повідомлень.

Створення вікон

Прикладна програма створює її вікна (включаючи головне) шляхом використання функцій*CreateWindow* і*CreateWindowEx* та забезпечуючи при цьому операційну систему необхідною для визначення атрибутів вікна інформацією. Win32 API надає інші функції (*DialogBox, CreateDialog* та *MessageBox)* для створення вікон спеціального призначення, таких, як діалогові вікна та вікна повідомлень.

Для створення вікна прикладна програма повинна вказати таку інформацію: клас вікна, його назву, стиль, вікно-власник, розмір та розміщення на екрані, ідентифікатор вікна-нащадка або ідентифікатор меню, та ідентифікатор екземпляра прикладної програми (якому належатиме вікно).

Приклад коду, який створює елементи керування TrackBar(повзунок), DateTimePick(календар) та StatusBar (статусний рядок) приведено у лістингу:…

```
   hListView:=CreateWindow (TRACKBAR_CLASS, ‘ ’,
   WS_VISIBLE or WS_CHILD) or TBS_HORS, 10, 10, 250, 40,
   hWindow, 0, HInatance, 0) ;
   hDateCreateWindow( DATET IMEPICK_CLASS, ‘ ’,
   WS_VISIBLE or WS_CHILD, 10 , 70, 100 , 20, hWindow, 0,
   HInetance, 0) ; hStatusBarCreateWindow(STATUSCASSNAME, ‘ ’,
WS_VISIBLE or  WS_CHILD, 0, 0, 0, 0, hWindow, 0, HInetance,
0);
…
```

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАДИ

"1. ProcessList" – отримання списку працюючих процесів.

"2. Thread TThread" – Використання класу для TThread для створення потоку.

"3. Threads SendMessage" – Узгодження роботи потоків за допомогою функції SendMessage.

"4. API Window" – Розробка вікна ПЗ з використанням WIN API.

"5. API ResDialog" – Діалогове вікно з елементами керування.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які бувають класи пріоритету процесу?
2. Якою функцією створюють новий процес?
3. Які головні дії виконує функція Windows API *CreateProcess*?
4. Яким є клас пріоритету процесу за замовчуванням?
5. У чому полягає специфіка використання класу пріоритету процесу REALTIME_PRIORITY_CLASS?
6. Як при створенні нового процессу вказати його пріоритет?
7. Як функція *CreateProcess* взаємодіє з підсистемами Windows?
8. Напишіть фрагмент програми для запуску прикладної програми, яка розташована за шляхом C:\Windows\notepad.exe. використайте для цього функцію *CreateProcess*.
9. Як при створенні процесу задати його пріоритет?
10. Які параметри функції *CreateProcess* є обов'язковими?
11. Як отримати дескриптор новоствореного процесу?
12. Чи можна при виклику функції *ShellExecute* задати пріоритет нового процесу?
13. Які є відмінності та спільні риси у роботі функцій *CreateProcess* та *WinExec*?
14. Які ви знаєте функції API для створення нового процесу?
15. Як програмно змінити пріоритет довільного процесу?
16. Як програмно змінити пріоритет власного процесу?

17. За допомогою якої функції отримують значення класу пріоритету процесу?
18. Яким чином можна програмно припинити виконання процесу?
19. Напишіть фрагмент програми, який запускає довільну прикладну програму й очікує на її завершення.
20. Для чого призначена функція API *OpenProcess*?
21. Як задати потрібний рівень доступу до процесу при виклику функції *OpenProcess*?
22. Для чого призначений дескриптор процесу?
23. Що таке ідентифікатор процесу?
24. Як можна отримати список ідентифікаторів запущених процесів?
25. Який системний процес має ідентифікатор 0?
26. Для чого використовують функцію *EnumProcessModules*?
27. Що таке «квант» в термінах ОС Windows?
28. У чому полягає принцип витісняю чого планування у керуванні потоками?
29. Яка частина ОС здійснює керування потоками?
30. Які є рівні пріоритету потоків?
31. Який взаємозв'язок між класами пріоритету процесу та рівнями пріоритету потоку?
32. Який рівень пріоритету потоку використовується за замовчуванням?
33. Що таке базовий пріоритет потоку?
34. Яка відмінність між базовим та поточним рівнями пріоритету?
35. Для чого Windows використовує поточне значення пріоритету?
36. У яких станах можуть перебувати потоки?
37. У чому полягає самостійне перемикання потоків?
38. Як відбувається витіснення одного потоку іншим?
39. Як відбувається перемикання потоків при завершенні кванта?
40. Що охоплює поняття контексту потоку?

41. Які переваги багатопотоковості у програмах?
42. Яка функція API призначення для створення нового потоку?
43. Як при створенні потоку задають його виконавчий код?
44. Як створити потік, не запускаючи його автоматично на виконання?
45. За допомогою яких функцій можна прочитати/ встановити пріоритет потоку?

Лабораторна робота №2 (семестр 2)

ТЕМА: Механізм керування пам’яттю

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

Механізм керування пам'яттю є однією з найважливіших складових операційної системи. У комп'ютерах IBM-сумісної архітектури застосунки разом з оброблюваними даними розташовуються в оперативній пам'яті. Таким чином, виконання будь-якого процесу обов'язково супроводжується операціями з пам'яттю.

Для реалізації схеми керування віртуальною пам'яттю, при якій кожен процес отримує власний закритий адресний простір, використовується диспетчервіртуальноїпам'яті(virtual memory manager).

Диспетчер пам'яті виконує два головних завдання: узгодження адрес віртуального адресного простору з наявними фізичними ресурсами пам'яті (трансляція віртуальних адрес) та керування використанням сторінкового файлу, який дозволяє компенсувати нестачу фізичної пам'яті.

Віртуальний адресний простір

Кожному процесу виділяється власний віртуальний адресний простір. Для 32-розрядних процесів його розмір становить 4 Гб (це максимальний обсяг, який може використовуватися при 32-бітній

адресації- 2^32). Для 64-розрядних процесів розмір адресного простору становить 16 екзабайт (2^64 байт). Потоки при виконанні отримують доступ тільки до пам'яті, яка належить його процесу. Пам'ять, відведена іншим процесам, прихована від потоку й недоступна для використання.

4 Гб адресного простору система ділить на дві приблизно однакові за обсягом частини. Перша частина надається у користування процесу, а друга резервується для системних потреб.

Віртуальний адресний простір кожного процесу розбивається на розділи, розмір і призначення яких залежать від версії Windows.

Щоб скористатися частиною доступного програмі адресного простору, в ньому потрібно виділити певні регіони за допомогою функції VirtualAlloc. Операцію виділення регіону називають резервуванням (reserving). При цьому система вирівнює початок регіону з урахуванням так званої гранулярності виділення пам'яті (allocation granularity), типове значення якої становить 64 Кб.

Коли зарезервований регіон адресного простору стає не потрібний, його слід повернути в загальні ресурси системи. Цю операцію називають вивільненням (releasing) регіону, вона виконується викликом функції VirtualFree.

Фізична пам'ять ділиться на сторінки, кожна з яких може мати окремі атрибути захисту(таблиця 1).

Атрибут захисту	Опис
PAGE_NOACCESS	Немає доступу. Спроби читання, запису або виконання вмісту пам'яті на цій сторінці викликають помилку доступу
PAGE_READONLY	Доступ лише для читання. Спроби запису або виконання вмісту пам'яті на цій сторінці викликають помилку доступу
PAGE_READWRITE	Доступ на читання та запис. Спроби виконання вмісту пам'яті на цій сторінці викликають помилку доступу
PAGE_EXECUTE	Лише виконання. Спроби читання або запису на цій сторінці викликають помилку доступу
PAGE_EXECUTE_READ	Читання та виконання. Спроби запису на цій сторінці викликають порушення прав доступу
PAGE_EXECUTE_READWRITE	Дозволено будь-який вид доступу
PAGE_WRITECOPY	Виконання вмісту пам'яті на цій сторінці викликає помилку доступу, запис приводить до того, що процесу надається "особиста" копія сторінки
PAGE_EXECUTE_WRITECOPY	На цій сторінці можливі будь-які операції, а спроба запису приводить до того, що процесу надається "особиста" копія сторінки

Таблиця 1. Атрибути захисту сторінок пам'яті

Припустимо, потрібно виділити регіон, починаючи з "позначки" 100 Мб в адресному просторі процесу. Тоді параметр lpvAddress повинен дорівнювати 104857600 (100-1024-1024). Якщо за цією адресою можна розмістити регіон необхідного розміру, функція зарезервує його й поверне відповідну адресу. Якщо ж вільного простору недостатньо, функція VirtualAlloc поверне nil. При цьому заданаадреса повинна належати розділу користувацького режиму процесу, інакше виникне помилка і VirtualAlloc також поверне nil.

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладі на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАД

“1. VirtualMemState” – дослідження структури адресного простору процесу.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовується диспетчер віртуальної пам’яті?
2. Який обсяг віртуального адресного простору на 32-бітних системах?
3. Опишіть структуру віртуального адресного простору процесу.
4. Який обсяг віртуальної пам’яті доступний користувацькому коду на 32-бітних системах?
5. Яке призначення розділу виявлення нульових вказівників?
6. Як прикладній програмі можна забезпечити максимальний обсяг доступної віртуальної пам’яті?
7. Опишіть схему використання віртуальної пам’яті у прикладній програмі.
8. Для чого використовується функція *VirtualAlloc*?

9. У чому полягає гранулярність виділення пам’яті? Які причини гранулярності?
10. Як можна виділити певний обсяг пам’яті за конкретно заданою адресою?
11. Як зарезервувати регіон адресного простору?
12. Яким чином можна отримати розмір сторінки пам’яті?
13. Як здійснюють проекцію фізичної пам’яті на віртуальну?
14. Як у програмі виділити 500МБ пам’яті та заповнити їх даними?
15. Як вивільняють раніше виділену пам'ять?
16. Для чого використовують сторінковий файл?
17. Чи може система працювати без сторінкового файлу?
18. Яка мінімальна порція даних має індивідуальні атрибути захисту?
19. Які атрибути захисту пам’яті ви знаєте?
20. Для чого використовують функцію *VitualFree*?
21. Як змінюють атрибути захисту регіону сторінок віртуальної пам’яті?
22. Як задати атрибути захисту певного байту віртуальної пам’яті?
23. Для чого призначена функція *VitrualQuery*?
24. Чи можна отримати інформацію про зайнятість віртуальної пам’яті стороннього процесу?
25. У чому полягають недоліки безпосереднього використання віртуальної пам’яті для невеликих масивів даних?
26. Яке призначення механізму куп?
27. Що таке купа?
28. Яким чином система забезпечує потокобезпечність куп?
29. Яка особливість середовища Delphi при роботі з купами?
30. Чи може Delphi-програма використовувати стандартний механізм куп?

Лабораторна робота №3 (семестр 2)

ТЕМА: Взаємодія між процесами IPC (Частина 1).

МЕТА: Отримати практичні навики в використанні функцій Win API. Навчитися організовувати коректну взаємодію між процесами застосовуючи механізми:

- **Повідомлення WM_COPYDATA;**
- **Відображення файлу у пам'ять.**

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

Повідомлення WM_COPYDATA – передає ділянку пам’яті одного процесу іншому за допомогою спеціального системного повідомлення.

Відображення файлу у пам'ять. Найчастіше спільний доступ кількох процесів до загальної ділянки пам’яті здійснюється з використанням механізму відображення файлів на пам’ять. Коли процес відображає файл на пам’ять він створює об’єкт відображення файлу, який дозволяє здійснювати доступ до вмісту файлу, так ніби файли розташовані в певному місці віртуального адресного простору процесу. Процес отримує можливість працювати з файлом, як з масивом , який зберігається в його пам’яті.

У зв'язку з великим обсягом інформації використовувати електронну документацію (погоджувати з лектором).

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАДИ

"1. IPC (WM_COPYDATA)" – IPC через повідомлення WM_COPYDATA.
"2. IPC (File mapping)" – IPC через запис у відображений файл.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовують механізми IPC?
2. Які способи міжпроцесної взаємодії ви знаєте?
3. Який спосіб IPC є найпростішим?
4. За допомогою якого повідомлення можна реалізувати передачу даних між процесами?
5. Чому для надсилання даних іншому процесу не можна використовувати функцію *PostMessage*?
6. Опишіть послідовність дій при взаємодії між процесами через WM_COPYDATA.
7. Напишіть фрагмент програми для передачі структури даних TData вікну з відомим дескриптором за допомогою WM_COPYDATA.
8. У чому полягає відображення файлу на пам'ять?

9. Для чого використовують механізм відображення файлів?
10. Чи можна реалізувати IPC за допомогою відображених файлів для процесів на різних машинах мережі?
11. Опишіть послідовність дій для реалізації IPC за допомогою відображення файлів?
12. Як відбувається створення об’єкта «файл»?
13. Як дисковий файл відображується у пам'ять?
14. Для чого використовується об’єкт ядра «канал»?
15. Які бувають канали?
16. Вкажіть послідовність дій для міжпроцесної взаємодії за допомогою іменованих каналів.
17. Який формат назви іменованого каналу?
18. Який процес є сервером каналу?
19. Чи можна реалізувати IPC для процесів на різних машинах за допомогою каналів?
20. Чому для роботи з каналами використовують ті ж функції, що і для роботи з файлами?
21. Які ключові особливості поштових скриньок?
22. Вкажіть формат назви поштової скриньки.
23. Опишіть послідовність дій для здійснення IPC за допомогою поштової скриньки.
24. Чи можна реалізувати передачу даних процесу на іншому комп’ютері за допомогою поштової скриньки?
25. Напишіть фрагмент програми для передачі даних іншому процесу через поштову скриньку PostBox.
26. Напишіть фрагмент програми для отримання даних від іншого процесу через поштову скриньку PostBox.
27. Як реалізувати двонаправлену передачу даних через поштову скриньку?

28. Як передавати повідомлення поштовим скринькам відразу на кількох комп’ютерах?
29. Що таке «сокет»?
30. У чому відмінність використання сокетів від інших механізмів IPC?
31. Як ідентифікується сокет?
32. Опишіть послідовність дій для передачі даних через сокети.
33. Для чого використовується порт сокета?
34. У чому різниця між синхронними та асинхронними сокетами?
35. Назвіть головні функції Winsock API для передачі даних через сокети.
36. Для чого використовують функцію *socket*?
37. Намалюйте блок-схему багатопотокового сервера, який отримує дані від клієнтів через сокет.
38. Для чого використовують функцію Winsock API *listen*?
39. Чи можна з одного процесу створити потік в іншому процесі?
40. Вкажіть послідовність дій, за допомогою яких можна завантажити сторонній код в адресний простір іншого процесу.

Лабораторна робота №4 (семестр 2)

ТЕМА: Взаємодія між процесами IPC (Частина 2).

МЕТА: Отримати практичні навики в використанні функцій Win API. Навчитися організовувати коректну взаємодію між процесами застосовуючи механізми:

– Іменований канал (named pipe);

– Поштові скриньки (mailslots).

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

Іменований канал (named pipe) - надзвичайно потужний інструмент взаємодії двох програм, які здійснюють обмін даними відповідно до концепції клієнт-сервер. Можуть використовуватись в ситуації коли програми працюють на різних комп'ютерах.

– Поштові скриньки (mailslots) можуть працювати в одному напрямку, тому при необхідності передавати дані в обох напрямках слід використовувати ддва поштові слоти. Перевага поштових слотів-можливість передати повідомлення через локальну мережу відразу кільком програмам за одну операцію.

У зв'язку з великим обсягом інформації використовувати електронну документацію (погоджувати з лектором).

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАДИ

" 3. IPC (Named pipes)" – IPC через іменований канал (named pipe).

" 4. IPC (Mailslots)" – IPC через поштові скриньки (mailslots).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовують механізми IPC?
2. Які способи міжпроцесної взаємодії ви знаєте?
3. Який спосіб IPC є найпростішим?
4. За допомогою якого повідомлення можна реалізувати передачу даних між процесами?
5. Чому для надсилання даних іншому процесу не можна використовувати функцію *PostMassage*?
6. Опишіть послідовність дій при взаємодії між процесами через WM_COPYDATA.
7. Напишіть фрагмент програми для передачі структури даних TData вікну з відомим дескриптором за допомогою WM_COPYDATA.
8. У чому полягає відображення файлу на пам'ять?

9. Для чого використовують механізми відображення файлів?
10. Чи можна реалізувати IPC за допомогою відображених файлів для процесів на різних машинах мережі?
11. Опишіть послідовність дій для реалізації IPC за допомогою відображення файлів.
12. Як відбувається створення об’єкта «файл»?
13. Як дисковий файл відображується у пам'ять?
14. Для чого використовується об’єкт ядра «канал»?
15. Які бувають канали?
16. Вкажіть послідовність дій для міжпроцесної взаємодії за допомогою іменованих каналів.
17. Який формат назви іменованого каналу?
18. Який процес є сервером каналу?
19. Чи можна реалізувати IPC для процесів на різних машинах за допомогою каналів?
20. Чому для роботи з каналами використовують ті ж функції, що і для роботи з файлами?
21. Які ключові особливості поштових скриньок?
22. Вкажіть формат назви поштової скриньки.
23. Опишіть послідовність дій для здійснення IPC за допомогою поштової скриньки.
24. Чи можна реалізувати передачу даних процесу на іншому комп’ютері за допомогою поштової скриньки?
25. Напишіть фрагмент програми для передачі даних іншому процесу через поштову скриньку PostBox.
26. Напишіть фрагмент програми для отримання даних від іншого процесу через поштову скриньку PostBox.
27. Як реалізувати двонаправлену передачу даних через поштову скриньку?

28. Як передавати повідомлення поштовим скринькам відразу на кількох комп’ютерах?
29. Що таке «сокет»?
30. У чому відмінність використання сокетів від інших механізмів IPC?
31. Як ідентифікується сокет?
32. Опишіть послідовність дій для передачі даних через сокети.
33. Для чого використовується порт сокета?
34. У чому різниця між синхронними та асинхронними сокетами?
35. Назвіть головні функції Winsock API для передачі даних через сокети.
36. Для чого використовують функцію *socket*?
37. Намалюйте блок-схему багатопотокового сервера, який отримує дані від клієнтів через сокет.
38. Для чого використовують функцію Winsock API *listen*?
39. Чи можна з одного процесу створити потік в іншому процесі?
40. Вкажіть послідовність дій, за допомогою яких можна завантажити сторонній код в адресний простір іншого процесу.

Лабораторна робота №5 (семестр 2)

ТЕМА: Взаємодія між процесами IPC (Частина 3).

МЕТА: Отримати практичні навики в використанні функцій Win API. Навчитися організовувати коректну взаємодію між процесами застосовуючи механізм:

– Сокети (Windows Sockets).

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Технологія «сокетів» (sockets-гніздо, роз’єм) розроблена для підтримки мережевої взаємодії програм.

У зв'язку з великим обсягом інформації використовувати електронну документацію (погоджувати з лектором).

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАДИ

"5. IPC (Sockets)" – IPC через зв’язок сокетів, клієнта і сервера.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовують механізми IPC?
2. Які способи міжпроцесної взаємодії ви знаєте?
3. Який спосіб IPC є найпростішим?
4. За допомогою якого повідомлення можна реалізувати передачу даних між процесами?
5. Чому для надсилання даних іншому процесу не можна використовувати функцію *PostMessage*?
6. Опишіть послідовність дій при взаємодії між процесами через WM_COPYDATA.
7. Напишіть фрагмент програми для передачі структури даних TData вікну з відомим дескриптором за допомогою WM_COPYDATA.
8. У чому полягає відображення файлу на пам'ять?
9. Для чого використовують механізм відображення файлів?
10. Чи можна реалізувати IPC за допомогою відображених файлів для процесів на різних машинах мережі?
11. Опишіть послідовність дій для реалізації IPC за допомогою відображення файлів?
12. Як відбувається створення об’єкта «файл»?
13. Як дисковий файл відображується у пам'ять?
14. Для чого використовується об’єкт ядра «канал»?
15. Які бувають канали?

16. Вкажіть послідовність дій для міжпроцесної взаємодії за допомогою іменованих каналів.
17. Який формат назви іменованого каналу?
18. Який процес є сервером каналу?
19. Чи можна реалізувати IPC для процесів на різних машинах за допомогою каналів?
20. Чому для роботи з каналами використовують ті ж функції, що і для роботи з файлами?
21. Які ключові особливості поштових скриньок?
22. Вкажіть формат назви поштової скриньки.
23. Опишіть послідовність дій для здійснення IPC за допомогою поштової скриньки.
24. Чи можна реалізувати передачу даних процесу на іншому комп'ютері за допомогою поштової скриньки?
25. Напишіть фрагмент програми для передачі даних іншому процесу через поштову скриньку PostBox.
26. Напишіть фрагмент програми для отримання даних від іншого процесу через поштову скриньку PostBox.
27. Як реалізувати двонаправлену передачу даних через поштову скриньку?
28. Як передавати повідомлення поштовим скринькам відразу на кількох комп'ютерах?
29. Що таке «сокет»?
30. У чому відмінність використання сокетів від інших механізмів IPC?
31. Як ідентифікується сокет?
32. Опишіть послідовність дій для передачі даних через сокети.
33. Для чого використовується порт сокета?
34. У чому різниця між синхронними та асинхронними сокетами?
35. Назвіть головні функції Winsock API для передачі даних через сокети.

36. Для чого використовують функцію *socket*?
37. Намалюйте блок-схему багатопотокового сервера, який отримує дані від клієнтів через сокет.
38. Для чого використовують функцію Winsock API *listen*?
39. Чи можна з одного процесу створити потік в іншому процесі?
40. Вкажіть послідовність дій, за допомогою яких можна завантажити сторонній код в адресний простір іншого процесу.

Лабораторна робота №6 (семестр 2)

ТЕМА: Системна взаємодія між процесами.

МЕТА: Отримати практичні навики в використанні функцій Win API. Навчитися організовувати коректну взаємодію між процесами застосовуючи механізм:

– Створення потоку у сторонньому процесі.

МЕТА: Отримати практичні навики використання функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

У зв'язку з великим обсягом інформації використовувати електронну документацію (погоджувати з лектором).

ЗАВДАННЯ

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові C++ чи C#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову C++ чи C#.

ПРИКЛАДИ

"6. RemoteThread" – Системна взаємодія між процесами через створення потоку у сторонньому процесі.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого використовують механізми IPC?
2. Які способи міжпроцесної взаємодії ви знаєте?
3. Який спосіб IPC є найпростішим?
4. За допомогою якого повідомлення можна реалізувати передачу даних між процесами?
5. Чому для надсилання даних іншому процесу не можна використовувати функцію *PostMessage*?
6. Опишіть послідовність дій при взаємодії між процесами через WM_COPYDATA.
7. Напишіть фрагмент програми для передачі структури даних TData вікну з відомим дескриптором за допомогою WM_COPYDATA.
8. У чому полягає відображення файлу на пам'ять?
9. Для чого використовують механізм відображення файлів?
10. Чи можна реалізувати IPC за допомогою відображених файлів для процесів на різних машинах мережі?
11. Опишіть послідовність дій для реалізації IPC за допомогою відображення файлів?
12. Як відбувається створення об’єкта «файл»?
13. Як дисковий файл відображується у пам'ять?
14. Для чого використовується об’єкт ядра «канал»?
15. Які бувають канали?
16. Вкажіть послідовність дій для міжпроцесної взаємодії за допомогою іменованих каналів.

17. Який формат назви іменованого каналу?
18. Який процес є сервером каналу?
19. Чи можна реалізувати IPC для процесів на різних машинах за допомогою каналів?
20. Чому для роботи з каналами використовують ті ж функції, що і для роботи з файлами?
21. Які ключові особливості поштових скриньок?
22. Вкажіть формат назви поштової скриньки.
23. Опишіть послідовність дій для здійснення IPC за допомогою поштової скриньки.
24. Чи можна реалізувати передачу даних процесу на іншому комп'ютері за допомогою поштової скриньки?
25. Напишіть фрагмент програми для передачі даних іншому процесу через поштову скриньку PostBox.
26. Напишіть фрагмент програми для отримання даних від іншого процесу через поштову скриньку PostBox.
27. Як реалізувати двонаправлену передачу даних через поштову скриньку?
28. Як передавати повідомлення поштовим скринькам відразу на кількох комп'ютерах?
29. Що таке «сокет»?
30. У чому відмінність використання сокетів від інших механізмів IPC?
31. Як ідентифікується сокет?
32. Опишіть послідовність дій для передачі даних через сокети.
33. Для чого використовується порт сокета?
34. У чому різниця між синхронними та асинхронними сокетами?
35. Назвіть головні функції Winsock API для передачі даних через сокети.
36. Для чого використовують функцію *socket*?

37. Намалюйте блок-схему багатопотокового сервера, який отримує дані від клієнтів через сокет.
38. Для чого використовують функцію Winsock API *listen*?
39. Чи можна з одного процесу створити потік в іншому процесі?
40. Вкажіть послідовність дій, за допомогою яких можна завантажити сторонній код в адресний простір іншого процесу.

Лабораторна робота №7 (семестр 2)

ТЕМА: ОСНОВИ ВИКОРИСТАННЯ СЛУЖБ ОС WINDOWS

МЕТА: Навчитися організовувати програми та коректно розв'язувати складні задачі. Отримати практичні навики в використанні функцій Win API.

ЗНАТИ: Основи програмування в ОС Windows.

ВМІТИ: Застосовувати API-функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ.

Служба - це програма зі спеціальною структурою, яка дозволяє ОС особливим чином керувати її роботою, даючи додаткові можливості адміністрування.

Разом з Windows поставляється значна кількість служб різноманітного
призначення. До них належать Task Sheduler (планувальник роботи програм),
Plug and Play (керує встановленням та налаштуванням пристроїв, відслідковує
зміни в апаратній конфігурації), Server (забезпечує підтримку RPC і сумісне
використання ресурсів комп'ютера) та багато інших. У вигляді служб розроблено велику кількість додаткового серверного програмного забезпечення, наприклад, СУБД Microsoft SQL Server, MySQL Server тощо.

Різниця між звичайною прикладною програмою та службою не є принциповою, тому існує спеціальна утиліта (srvany.exe), яка дозволяє запускати існуючу програму у вигляді служби. Але при цьому в повній мірі не

забезпечуються можливості щодо адміністрування такої служби.

Так як служба призначена для роботи на серверній машині, яка не використовується в ролі робочої станції, то в служб найчастіше відсутній користувацький інтерфейс. А операції щодо адміністрування та налаштування

служб здійснюють за допомогою спеціалізованого системного та прикладного програмного забезпечення.

Механізм служб у Windows забезпечується взаємодією таких головних компонентів, як диспетчер керування службами, саме служб та програм для керування службами.

Диспетчеркеруванняслужбами (ServiceControlManager, SCM)- це спеціалізований компонент ОС (він представлений виконавчим файлом Services.exe), який автоматично запускається під час запуску Windows і працює

Програма керування службою (ServiceControlProgram, SCP) – це звичайна прикладна програма, яка має користувацький інтерфейс для налаштування, запуску, зупинки, призупинки, продовження роботи та виконання інших функцій керування службою. Така програма викликає спеціальні функції Windows, які дозволяють їй взаємодіяти з SCM. SCP не може напряму взаємодіяти зі службою, передача даних службі та їх отримання від неї здійснюються через посередництво SCM.

Встановлення та видалення служби

Для встановлення служби простого запуску її виконавчого модуля недостатньо: потрібно дати явну вказівку диспетчеру керування службами про необхідність її встановлення. Спочатку треба встановити зв'язок

&SCM за допомогою функції OpenSCManager, яка з'єднується з диспетчером на заданій машині та відкриває його базу даних.

Для видалення (деінсталяції) служби з бази даних диспетчера керування службами використовують функцію DeleteService.

Після закінчення роботи зі службою її дескриптор (як і дескриптор бази даних SCM) необхідно закрити викликом CloseServiceHandle

ЗАВДАННЯ

Використовуючи наведені приклади – "GetServerInfoVCL", "Service GetServerInfoSocket", "ApplicationA" та наявну електронну документацію по службам ОС WINDOWS, написати службу яка з інтервалом 5 секунд передає у мережу наступні дані:

– Ім’я сервера (*GetComputerName*);

– Ім’я поточного користувача ПК (*GetUserName*);

– Назва та версія ОС(*GetVersionEx*).

Приклади.

«1. GetServerInfoSocket» – Служба для розсилання даних про параметри сервера.

«2. ApplicationA» – Отримувач даних з сервера.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке «служба» в термінах ОС Windows?
2. Яка відмінність служби від звичайної прикладної програми?
3. Назвіть відомі приклади реалізації служб.
4. Які переваги надає використання служб?
5. Назвіть головні компоненти механізму служб у Windows.
6. для чого призначений диспетчер керування службами?
7. Яке призначення програми керування службою?
8. Скільки служб може містити один виконавчий файл?

9. Як стандартними засобами Windows можна проглянути список встановлених служб?
10. В якому підрозділі реєстру зберігаються дані про служби?
11. Опишіть програмну структуру служби.
12. Для чого призначена функція головного потоку виконавчого файлу служби?
13. Яке призначення функції потоку служби?
14. Як програмно забезпечується взаємодія служби з диспетчером керування службами?
15. Як задають список служб, наявних у виконавчому файлі?
16. Для чого використовують функцію *StartServiceCtrlDispatcher*?
17. Який прототип функції потоку служби?
18. У якій вітці реєстру слід зберігати параметри служб?
19. Для чого призначена функція *RegisterServiceCtrlHandler*?
20. Якою функцією можна змінити стан служби?
21. Яке призначення структури SERVICE_STATUS?
22. Як здійснюється програмне керування службою?
23. Як здійснюють встановлення та видалення служб?
24. Який результат повертає функція *OpenSCManager*?
25. Для чого призначена функція *CreateService*?
26. Якою функцією запускають службу на виконання?
27. Як можна отримати дескриптор вже запущеної служби?
28. Напишіть код, який запускає службу з відомим дескриптором.
29. Які дії виконує функція *DeleteService*?
30. Вкажіть послідовність дій при розробці служби засобами Windows API.
31. Який клас у Delphi реалізує функціональність служб?
32. Приведіть послідовність дій для створення служби у Delphi.

33. Як реалізують функцію потоку служби за допомогою класу TService?
34. Назвіть головні властивості та методи класу TService.
35. Які переваги та недоліки використання автоматизаного методу розробки служби за допомогою класу TService?

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Коноваленко І.В., Федорів П.С. Системне програмування у Windows з прикладами на Delphi. Навчальний посібник для технічних спеціальностей вищих навчальних закладів. – Тернопіль:ТНТУ ім. І. Пулюя, 2012. -320с.
2. Руссинович М. и Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, WindowsXP и Windows 2000. Мастер_класс. / Пер. с англ. — 4_е изд. — М.: Издательство «Русская Редакция»; СПб.: Питер, 2008. — 992 стр.:
3. Побегайло А.П. Системное программирование в Windows.- СПб.: БХВ-Петербург, 2006.- 1056с.: ил.
4. Щупак Ю.А. Win32 API. Эффективная разработка приложений.- СПб.: Питер, 2007.- 572с.: ил.
5. Гери Неббет Справочник по базовым функциям API Windows NT/2000. Издательский дом «Вильямс», 2002.-528 с.:ил.-Парал. тит. англ.
6. Эндрю Кровчик, Винод Кумар, Номан Лагари, Аджит Мунгале, Кристиан Нагел, Тим Паркер, Шриниваса Шивакумар. .NET. Сетевое программирование для профессионалов. Иддательский дом «Лори». 2005.