Лабораторна робота №9

Тема. Організація взаємодії між процесами

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

Теоретичні відомості

Існує досить великий клас засобів ОС, за допомогою яких забезпечується взаємна синхронізація процесів і потоків. Потреба в синхронізації потоків виникає тільки в мультипрограмній ОС і залежить від спільного використання апаратних та інформаційних ресурсів обчислювальної системи. Синхронізація потрібна для запобігання перегонам та безвиході під час обміну даними між потоками,поділу даних, доступу до процесора і пристроїв введення-виведення.У багатьох ОС ці засоби називаються засобами міжпроцесної взаємодії – Inter Process Communications (IPC), що відображає історичну первинність поняття процес відносно поняття потік. Зазвичай до засобів IPC належать не тільки засоби міжпроцесної синхронізації, але й засоби обміну даними. Будь-яка взаємодія процесів або потоків залежить від їх синхронізації, яка полягає в узгодженні їх швидкостей через припинення потоку до настання деякої події й подальшої його активізації під час настання цієї події. Синхронізація лежить в основі будь-якої взаємодії потоків, незалежно від того, чи пов’язана ця взаємодія з розподілом ресурсів або з обміном даними. Наприклад, потік-одержувач повинен звертатися за даними тільки після того, як вони поміщені в буфер потоком-відправником. Якщо ж потік-одержувач звернувся до даних до моменту їх надходження в буфер, то він має бути припинений. Синхронізація також потрібна у разі спільного використання апаратних ресурсів. Наприклад, коли активному потоку потрібен доступ до послідовного порту, а з цим портом у монопольному режимі працює інший потік, який перебуває у стані очікування, то ОС припиняє активний потік і не активізує його доти, доки потрібний йому порт не звільниться. Часто потрібна також синхронізація з подіями, які не належать до обчислювальної системи, наприклад реакція на натискання комбінації клавіш Ctrl+C. Для синхронізації потоків прикладних програм програміст може використовувати як власні засоби та прийоми синхронізації, так і засоби ОС. Наприклад, два потоки одного прикладного процесу можуть координувати свою роботу за допомогою доступної для них обох глобальної логічної змінної, яка набуває значення одиниці при здійсненні деякої події, наприклад вироблення одним потоком даних, потрібних для продовження роботи іншого. Однак у багатьох випадках більш ефективними або навіть єдино можливими є засоби синхронізації, що надаються ОС у формі системних викликів. Так, потоки, що належать різним процесам, не мають можливості втручатися будь-яким чином у

роботу один одного. Без посередництва ОС вони не можуть призупинити один одного або сповістити про подію, що відбулася. Засоби синхронізації використовуються ОС не тільки для синхронізації прикладних процесів, але й для її внутрішніх потреб. Зазвичай розробники ОС надають у розпорядження прикладних і системних програмістів широкий спектр засобів синхронізації. Ці засоби можуть утворювати ієрархію, коли на основі більш простих засобів будуються більш складні, а також бути функціонально спеціалізованими, наприклад засоби для синхронізації потоків одного процесу, засоби для синхронізації потоків різних процесів під час обміну даними і т. ін. Часто функціональні можливості різних системних викликів синхронізації перекриваються, тому для вирішення одного завдання програміст може скористатися кількома викликами залежно від особистих переваг.

Ситуації, коли процесам доводиться взаємодіяти:

– передавання інформації від одного процесу до іншого;

– контроль над діяльністю процесів (наприклад, коли вони змагаються за один ресурс);

– узгодження дій процесів (наприклад, коли один процес доставляє дані, а інший їх виводить на друк. Якщо узгодженості не буде, то другий процес може почати друк раніше, ніж надійдуть дані).

Два останні випадки стосуються і потоків. У першому випадку потоки не мають проблем, тому що вони використовують загальний адресний простір.

Існує дві основні моделі міжпроцесорної комунікації: спільна пам'ять та передача повідомлень. У моделі спільної пам’яті встановлюється область пам’яті, яка поділяється процесами співпраці. Потім процеси можуть обмінюватися інформацією, читаючи та записуючи дані в спільний регіон. У моделі передачі повідомлень, спілкування відбувається за допомогою повідомлень, що обмінюються між взаємодіючими процесами.

Передавання інформації від одного процесу до іншого. Інформація може передаватися кількома способами:

– колективна пам’ять;

– канали (труби)– це псевдофайл, який один процес записує, а інший зчитує.

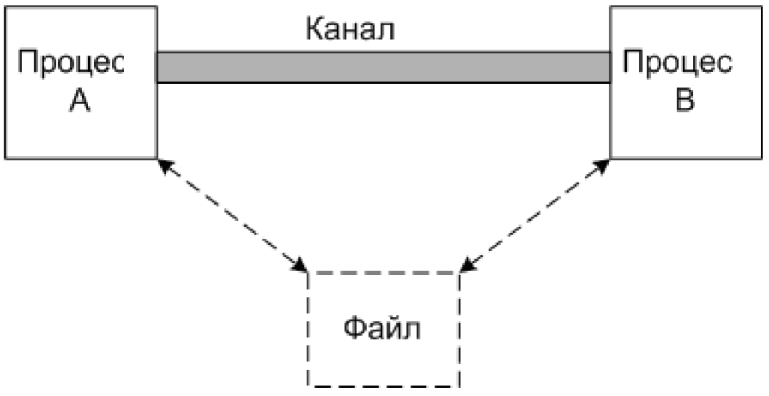


Схема для каналу

– сокети – підтримувані ядром механізми, що приховують особливості середовища і дозволяють взаємодіяти процесам, як на одному комп’ютері, так і в мережі.

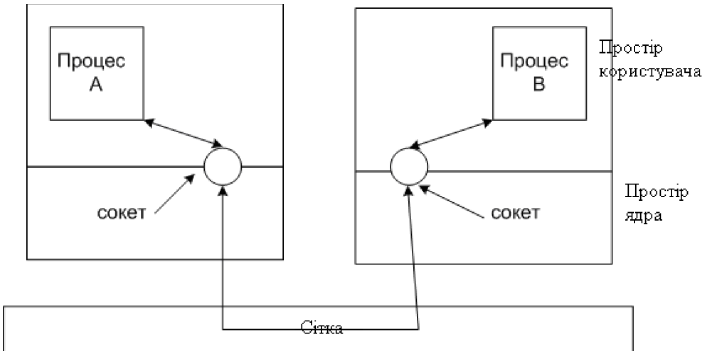


Схема для сокетів

– поштові скриньки (тільки у Windows) – однонапрямлені з можливістю широкомовної розсилки;
– виклик віддаленої процедури – процес А може викликати процедуру в процесі В і отримувати назад дані.

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

чекати нових товарів, але виробник завжди може випускати нові товари. Обмежений буфер передбачає фіксований розмір буфера. У цьому випадку споживач повинен чекати, якщо буфер порожній, а виробник повинен чекати, якщо буфер заповнений. Давайте докладніше розглянемо, як обмежений буфер ілюструє міжпроцесовий зв’язок за допомогою спільної пам'яті. Наступні змінні знаходяться в області пам’яті, поділеної процесорами виробника та споживача:

```
#define BUFFER SIZE 10
typedef struct { ... } item;
 item buffer[BUFFER SIZE];
int in = 0;
int out = 0;

item next consumed;
while (true) { while (in == out) ; /* do nothing */
next consumed = buffer[out]; out = (out + 1) % BUFFER SIZE;
/* consume the item in next consumed */
}
```

POSIX спільна пам'ять організовується за допомогою файлів, відображених у пам'яті, які пов'язують область спільної пам'яті з файлом. Спочатку A process повинен створити об'єкт спільної пам’яті за допомогою системного виклику shm_open () таким чином:

```
fd = shm_open(ім'я, O_CREAT | O_RDWR, 0666);
```

Перші параметри визначають ім'я об'єкта спільної пам’яті. Процеси, які бажають отримати доступ до цієї спільної пам’яті, повинні посилатися на об’єкт цим ім’ям. Подальші параметри визначають, що це пам'ять, що створюється в об'єкті, створюється пристосованим текстовим текстом (O_CREAT) і тим, що об'єктивуються для читання та запису (O_RDWR). Останній параметр встановлює дозволи на доступ до файлів об'єкта спільної пам'яті. Успішний виклик shm_open() повертає дескриптор цілого файлу для об'єкта спільної пам’яті. Після встановлення об'єкта функція ftruncate() використовується для налаштування розміру об'єкта в байтах. Виклик

```
ftruncate (fd, 4096);
```

встановлює розмір об'єкта 4096 байт.

Нарешті, mmap() функціонує встановлення відображення файлу у пам’ять, що містить розділений об'єкт пам'яті.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
 #include <sys/shm.h>
```

```
#include <sys/stat.h>
#include <sys/mman.h>
int main() {
 const int SIZE = 4096;
const char *name = "OS";
const char *message 0 = "Hello";
const char *message 1 = "World!";
/* shared memory file descriptor */ int fd;
/* pointer to shared memory obect */ char *ptr;
/* create the shared memory object */ fd = shm open(name,O CREAT | O RDWR,0666);
/* configure the size of the shared memory object */ ftruncate(fd, SIZE);
/* memory map the shared memory object */ ptr = (char *) mmap(0, SIZE, PROT READ | PROT WRITE, MAP SHARED, fd, 0);
/* write to the shared memory object */ sprintf(ptr,"%s",message 0); ptr += strlen(message 0); sprintf(ptr,"%s",message 1);
 ptr += strlen(message 1);
return 0;
}
```

```
#include <stdio.h> #include <stdlib.h> #include <fcntl.h> #include <sys/shm.h> #include <sys/stat.h>
#include <sys/mman.h>
int main() {
 /* the size (in bytes) of shared memory object */ const int SIZE = 4096;
 /* name of the shared memory object */ const char *name = "OS";
/* shared memory file descriptor */ int fd;
/* pointer to shared memory obect */ char *ptr;
/* open the shared memory object */ fd = shm open(name, O RDONLY, 0666);
/* memory map the shared memory object */ ptr = (char *) mmap(0, SIZE, PROT READ | PROT WRITE, MAP SHARED, fd, 0);
/* read from the shared memory object */ printf("%s",(char *)ptr);
/* remove the shared memory object */ shm unlink(name);
return 0;
}
```

Для роботи з пам'яттю, що розділяється використовуються системні виклики:

· shmget створює новий сегмент розподіленої пам'яті або знаходить існуючий сегмент з тим самим ключем;

· shmat підключає сегмент із зазначеним вказівником до віртуальної пам'яті процесу, що здійснює звернення;
· shmdt відключає від віртуальної пам'яті раніше підключений до неї сегмент із зазначеною віртуальною адресою початку;
· shmctl служить для управління різноманітними параметрами, пов'язаними з існуючим сегментом.
Прототипи перерахованих системних викликів описані в файлах
#include <sys / ipc.h>
#include <sys / shm.h>
Після того як сегмент розподіленої пам'яті підключений до віртуальної пам'яті процесу, цей процес може звертатися до відповідних елементів пам'яті з використанням звичайних машинних команд читання і запису.
Системний виклик
int shmid = shmget (key_t key, size_t size, int flag)
на підставі параметру size визначає бажаний розмір сегменту в байтах. Якщо в таблиці розподіленої пам'яті знаходиться елемент, що містить заданий ключ, і права доступу не суперечать поточним характеристикам процесу, що здійснює звернення, то значенням системного виклику є ідентифікатор існуючого сегменту, причому параметр size повинен бути в цьому випадку рівним 0. В іншому випадку створюється новий сегмент з розміром не менше встановленого в системі мінімального розміру сегменту розподіленої пам'яті і не більше встановленого максимального розміру. Живучість об'єктів розподіленої пам'яті визначається живучістю ядра. Створення сегменту не означає негайного виділення під нього основної пам'яті, і ця дія відкладається до виконання першого системного виклику підключення сегменту до віртуальної пам'яті деякого процесу. Прапори IPCCREAT і IPCEXCL аналогічні розглянутим вище.
Підключення сегменту до віртуальної пам'яті виконується шляхом звернення до системного виклику shmat
void * virtaddr = shmat (intshmid, void * daddr, int flags)
Параметр shmid – це раніше отриманий ідентифікатор сегменту, а daddr – бажана процесом віртуальна адреса, яка повинна відповідати початку сегменту в віртуальній пам'яті.
Значенням системного виклику є фактична віртуальна адреса початку сегменту. Якщо значенням daddr є NULL, ядро вибирає найбільш зручну віртуальну адресу початку сегмента. Прапори системного виклику shmat наведені нижче в таблиці.
Для відключення сегменту від віртуальної пам'яті використовується системний виклик shmdt:
int shmdt (* daddr)
де daddr – це віртуальна адреса початку сегменту у віртуальній пам'яті, раніше отриманий від системного виклику shmat.
Системний виклик shmctl
int shmctl (int shmid, int command, struct shmid_ds * shrn_stat)

по синтаксису і призначенню аналогічний msgctl.

Файл in.c

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <error.h>
#include <fcntl.h>
#define SVSHM_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
#define STRING_SIZE 500
int main(int argc, char **argv)
{
if (argc != 2)
{
printf("usage: shmwrite <path_to_file>\n");
return 0;
}
printf ("type the message:\n");
char s[STRING_SIZE];
size_t length = 0;
int i = 0;
char c;
while (((c = getchar()) != '\n')&&(i < STRING_SIZE - 1))
{
s[i++] = c;
length++;
}
s[i] ='\0';
int oflag = SVSHM_MODE|IPC_CREAT;
int id = shmget(ftok(argv[1], 0), length, oflag);
unsigned char *ptr = (unsigned char*) shmat(id, NULL, 0);
struct shmid_ds buff;
shmctl(id, IPC_STAT, &buff);
for (i = 0; i < length; i++)
*ptr++ = s[i];
return 0;
}
```

Файл out.c

```
#include <stdio.h>
#include <string.h>
```

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <error.h>
#include <fcntl.h>
#define SVSHM_MODE ( S_IRUSR| S_IWUSR | S_IRGRP | S_IROTH )
int main(int argc, char **argv)
{
if (argc != 2)
{
printf("usage: shmwrite <path_to_file>\n");
return 0;
}
int id = shmget(ftok(argv[1], 0), 0, SVSHM_MODE);
unsigned char *ptr = (unsigned char*) shmat(id, NULL, 0);
struct shmid_ds buff;
shmctl(id, IPC_STAT, &buff);
int i;
for (i = 0; i < buff.shm_segsz; i++)
printf ("%c",(unsigned char) *ptr++);
printf("\n");
shmctl(id, IPC_RMID, NULL);
return 0;
}
```

У WINDOWS підтримуються наступні механізми міжпроцесної взаємодії:

- Clipboard
- COM
- Data Copy
- DDE
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

Clipboard

Буфер обміну є центральним сховищем для обміну даними між додатками. Коли користувач виконує операцію з вирізанням або копіюванням у програмі, програма розміщує вибрані дані в буфер обміну в одному або декількох стандартних або визначених додатком форматах. Потім будь-яка інша програма може отримати дані з буфера обміну, вибираючи з наявних форматів, які він

розуміє. Буфер обміну - це дуже вільно пов'язаний обмінний носій, де додаткам потрібно лише узгодити формат даних. Програми можуть перебувати на одному комп’ютері або на різних комп’ютерах у мережі. Усі програми повинні підтримувати буфер обміну для тих форматів даних, які вони розуміють. Наприклад, текстовий редактор або текстовий процесор повинен принаймні мати можливість створювати та приймати дані буфера обміну в чистому текстовому форматі.

COM

Програми, що використовують OLE, керують складовими документами, тобто документами, що складаються з даних із різних програм. OLE надає послуги, які спрощують додатки для виклику інших програм для редагування даних. Наприклад, текстовий процесор, який використовує OLE, може вбудувати графік з електронної таблиці. Користувач може запустити електронну таблицю автоматично з текстового процесора, вибравши вбудовану діаграму для редагування. OLE піклується про запуск електронної таблиці та презентацію графіка для редагування. Коли користувач вийде з електронної таблиці, графік буде оновлений у вихідному документі текстового редактора. Електронна таблиця представляється розширенням текстового процесора. Основою OLE є компонентна модель об'єкта (COM). Програмний компонент, що використовує COM, може спілкуватися з найрізноманітнішими іншими компонентами, навіть тими, які ще не написані. Компоненти взаємодіють як об'єкти, так і клієнти. Розподілений COM розширює модель програмування COM так, що вона працює в мережі. OLE підтримує складені документи та дозволяє додатку включати вбудовані або пов’язані дані, які при виборі автоматично запускають іншу програму для редагування даних. Це дає змогу розширити програму будь-яким іншим додатком, що використовує OLE. Об'єкти COM забезпечують доступ до даних об'єкта через один або кілька наборів пов'язаних функцій, відомих як інтерфейси.

Data Copy

Копія даних дозволяє додатку надсилати інформацію до іншої програми за допомогою повідомлення WM_COPYDATA. Цей метод вимагає співпраці між відправляючою заявою та приймаючою заявою. Заявка, що отримує, повинна знати формат інформації та мати можливість ідентифікувати відправника. Програма, що надсилає, не може змінювати пам'ять, на яку посилаються жодні вказівники. Копія даних може бути використана для швидкого надсилання інформації в іншу програму за допомогою обміну повідомленнями Windows.

DDE

DDE - це протокол, який дозволяє програмам обмінюватися даними в різних форматах. Програми можуть використовувати DDE для одноразового обміну даними або для постійних обмінів, коли додатки оновлюють один одного, коли нові дані стають доступними. Формати даних, використовувані DDE, такі ж, як і в буфері обміну. DDE можна розглядати як розширення механізму буфера обміну.

Буфер обміну майже завжди використовується для одноразової відповіді на команду користувача, наприклад, для вибору команди меню Вставити. DDE також зазвичай ініціюється командою користувача, але часто продовжує функціонувати без подальшої взаємодії з користувачем. Ви також можете визначити власні формати даних DDE для IPC спеціального призначення між додатками з більш щільно пов'язаними вимогами до зв'язку. Обмін DDE може відбуватися між програмами, що працюють на одному комп’ютері або на різних комп'ютерах у мережі. DDE не настільки ефективний, як новіші технології. Однак ви все одно можете використовувати DDE, якщо інші механізми IPC не підходять або якщо ви повинні взаємодіяти з існуючим додатком, який підтримує лише DDE.

File Mapping

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

Mailslots

Електронна пошта забезпечує односторонній зв'язок. Будь-який процес, який створює розсилку, є сервером розсилки. Інші процеси, звані клієнтами розсилки, надсилають повідомлення на сервер розсилки, надсилаючи повідомлення на свій розсилку. Вхідні повідомлення завжди додаються до розсилки. Електронна пошта зберігає повідомлення, поки сервер пошти не прочитає їх. Процес може бути як сервером розсилки, так і клієнтом розсилки, тому двосторонній зв'язок можливий за допомогою декількох розсилок. Клієнт розсилки може надіслати повідомлення на свій місцевий комп'ютер, на інший комп'ютер або на всі повідомлення з однаковим іменем на всіх комп’ютерах у визначеному мережевому домені.

Повідомлення, що транслюються на всі поштові розсилки в домені, не можуть перевищувати 400 байт, тоді як повідомлення, що надсилаються на один поштовий слот, обмежені лише максимальним розміром повідомлень, визначеним сервером пошти при створенні розсилки. Поштові розсилки пропонують простий спосіб для додатків надсилати та отримувати короткі повідомлення. Вони також забезпечують можливість трансляції повідомлень на всіх комп’ютерах у мережевому домені.

Pipes

Існує два типи труб для двостороннього зв'язку: анонімні труби та названі труби. Анонімні труби дозволяють пов'язаним процесам передавати інформацію один одному. Як правило, анонімна труба використовується для перенаправлення стандартного вводу або виводу дочірнього процесу, щоб він міг обмінюватися даними зі своїм батьківським процесом. Для обміну даними в обох напрямках (дуплексна робота) необхідно створити дві анонімні труби. Батьківський процес записує дані в одну трубку за допомогою своєї ручки запису, тоді як дочірній процес зчитує дані з цієї трубки за допомогою своєї ручки читання. Аналогічно, дочірній процес записує дані в іншу трубку і батьківський процес зчитує з нього. Анонімні труби не можуть використовуватися через мережу, а також не можуть використовуватися між спорідненими процесами. Названі труби використовуються для передачі даних між процесами, які не пов'язані між собою, і між процесами на різних комп'ютерах. Як правило, сервер з іменованим каналом створює іменовану трубу з добре відомим ім'ям або ім'ям, яке слід повідомити своїм клієнтам. Клієнтський процес з іменованою трубкою, який знає ім'я труби, може відкрити інший його кінець, за умови обмежень доступу, визначених сервером процес з іменованим каналом. Після того як і сервер, і клієнт підключилися до труби, вони можуть обмінюватися даними, виконуючи операції читання і запису на трубі. Анонімні канали забезпечують ефективний спосіб перенаправлення стандартного вводу або виводу на дочірні процеси на одному комп’ютері. Названі труби забезпечують простий інтерфейс програмування для передачі даних між двома процесами, незалежно від того, чи вони перебувають на одному комп’ютері чи по мережі.

RPC

RPC дозволяє програмам дистанційно викликати функції. Тому RPC робить IPC таким же простим, як і виклик функції. RPC працює між процесами на одному комп'ютері або на різних комп'ютерах у мережі. RPC, що надається Windows, відповідає розподіленому обчислювальному середовищу Open Source Foundation (OSF) (DCE). Це означає, що програми, які використовують RPC, можуть спілкуватися з програмами, що працюють з іншими операційними системами, що підтримують DCE. RPC автоматично підтримує перетворення даних для обліку різних архітектурних апаратних засобів та для впорядкування байтів між різними середовищами. Клієнти та сервери RPC тісно пов'язані, але все ще підтримують високу продуктивність. Система широко використовує RPC для полегшення

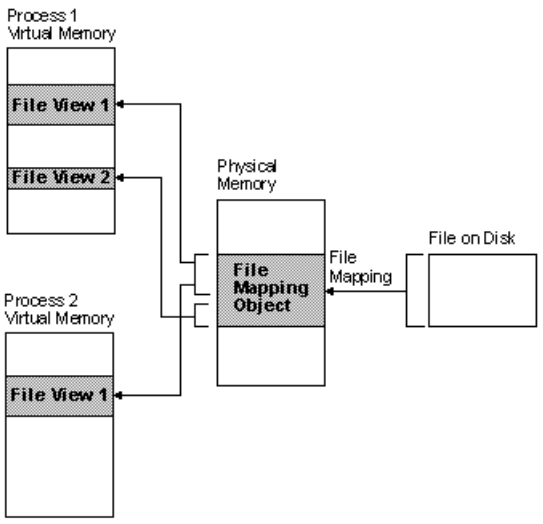
відносин клієнт / сервер між різними частинами операційної системи. RPC - це інтерфейс на рівні функцій, який підтримує автоматичне перетворення даних та зв'язок з іншими операційними системами. Використовуючи RPC, ви можете створювати високоефективні, щільно поєднані розподілені програми.

Windows Sockets

Windows Sockets - це незалежний від протоколу інтерфейс. Він використовує переваги комунікаційних можливостей базових протоколів. У Windows Sockets 2 дескриптор сокету може додатково використовуватися як файловий дескриптор зі стандартними функціями вводу / виводу файлів. Сокети Windows засновані на сокетах, вперше популяризованих Berkeley Software Distribution (BSD). Додаток, що використовує Windows Sockets, може спілкуватися з іншою реалізацією сокета в інших типах систем. Однак не всі постачальники транспортних послуг підтримують усі доступні варіанти. Windows Sockets - це незалежний від протоколу інтерфейс, здатний підтримувати поточні та нові можливості мереж.

File Mapping

Відображення файлів (File mapping)- це об'єднання вмісту файлу з частиною віртуального адресного простору процесу. Система створює об’єкт відображення файлів (file mapping object) (також відомий як section object) для підтримки цієї асоціації. file view - це частина віртуального адресного простору, яку використовує процес для доступу до вмісту файлу. Відображення файлів дозволяє процесу використовувати як випадковий вхід і вихід (I/O) , так і послідовні (I/O) . Це також дозволяє ефективно працювати з великим файлом даних, таким як база даних, без необхідності зіставляти весь файл у пам'ять. Кілька процесів також можуть використовувати файли, зібрані в пам'яті, для обміну даними. Процеси зчитують та записують у file view використовуючи вказівники так само, як і з динамічно виділеною пам'яттю. Використання відображення файлів підвищує ефективність, оскільки файл знаходиться на диску, але file view знаходиться в пам'яті. Процеси також можуть управляти file view за допомогою функції VirtualProtect.



Файлом на диску може бути будь-який файл, який ви хочете зіставити в пам'ять, або це system page file.. Відображення файлів може складатися з усієї або лише частини файлу. Він підтримується файлом на диску. Це означає, що коли система замінює сторінки об’єкта відображення файлів, будь-які зміни, внесені до об’єкта відображення файлів, записуються у файл. Коли сторінки об’єкта відображення файлів повертаються назад, вони відновлюються з файлу. file view може складатися з усієї або лише частини об'єкта відображення файлів. Процес маніпулює файлом через file view. Процес може створити кілька file views для об’єкта відображення файлів. file views, створені кожним процесом, знаходяться у віртуальному адресному просторі цього процесу. Коли процес потребує даних з частини файлу, відмінного від того, що знаходиться у поточному поданні файлу, він може скасувати карту поточного подання файлу, а потім створити новий вид файлу. Коли багато процесів використовують один і той же об’єкт відображення файлів для створення представлень для локального файлу, дані є когерентними. Тобто, перегляди містять однакові копії файлу на диску. Файл не може перебувати на віддаленому комп'ютері, якщо ви хочете ділитися пам'яттю між декількома процесами.
Перший крок у відображенні файлу - це відкрити файл, викликавши функцію CreateFile. Вона створює або відкриває файл або пристрій вводу / виводу. Найчастіше використовуються такі пристрої вводу / виводу: файл, file stream, каталог, фізичний диск,volume, консольний буфер, стрічковий накопичувач, ресурс зв'язку, mailslot та pipe. Функція повертає ручку, яка може бути використана для доступу до файлу чи пристрою для різних типів вводу-виводу в залежності від файлу чи пристрою та вказаних прапорів та атрибутів.

HANDLE CreateFileA(
LPCSTR lpFileName, // Назва файлу чи пристрою, який потрібно створити або відкрити.
DWORD dwDesiredAccess, // доступ до файлу чи пристрою GENERIC_READ, GENERIC_WRITE,
DWORD dwShareMode, // режим спільного доступу до файлу чи пристрою, який можна читати, записувати, обидва, видаляти, всі ці або жодні Якщо цей параметр дорівнює нулю, і CreateFile вдалий, файл або пристрій не можна спільно використовувати і не можна відкрити знову, поки дескриптор файлу чи пристрою не закритий. Щоб увімкнути процес спільного доступу до файлу або пристрою, коли інший процес відкрив файл або пристрій, використовуйте сумісну комбінацію одного або декількох із наведених нижче значень.
FILE_SHARE_DELETE \ FILE_SHARE_READ \ FILE_SHARE_WRITE
LPSECURITY_ATTRIBUTES lpSecurityAttributes,// вказівник на структуру SECURITY_ATTRIBUTES

DWORD dwCreationDisposition, // Дія, яку слід здійснити над файлом чи пристроєм, який існує або не існує CREATE_ALWAYS \ CREATE_ALWAYS \ OPEN_ALWAYS \ OPEN_EXISTING \ TRUNCATE_EXISTING
DWORD dwFlagsAndAttributes, //
Атрибути файлів або пристроїв та прапори, за замовчуванням FILE_ATTRIBUTE_NORMAL
HANDLE hTemplateFile);// дескриптор файлу шаблону з правом доступу GENERIC_READ. Файл шаблону надає атрибути файлу та розширені атрибути для файлу, який створюється.Цей параметр може бути NULL.

Щоб переконатися, що інші процеси не можуть записати у відповідну частину файлу, слід відкрити файл з ексклюзивним доступом. Крім того, дескриптор файлу повинна залишатися відкритою, поки процес більше не потребуватиме об'єкт відображення файлів. Найпростіший спосіб отримати ексклюзивний доступ - це вказати нуль у параметрі fdwShareMode у CreateFile. дескриптор, повернена CreateFile, використовується функцією CreateFileMapping для створення об’єкта відображення файлів.

HANDLE CreateFileMappingA(
HANDLE hFile, //дескриптор файлу
LPSECURITY_ATTRIBUTES lpFileMappingAttributes, //вказівник на //структуру SECURITY_ATTRIBUTES, що визначає чи повернений дескриптор може //успадковуватись дочірніми процесами
DWORD flProtect, //Визначає захист сторінки об'єкта відображення файлів. Усі //відображені види об’єкта повинні бути сумісні з цим захистом. PAGE_EXECUTE_READ \ //PAGE_EXECUTE_READWRITE \ PAGE_EXECUTE_WRITECOPY \ PAGE_READONLY \ PAGE_READWRITE \ //PAGE_WRITECOPY
Додаток може вказати один або декілька наступних атрибутів для об’єкта відображення файлів, поєднуючи їх з одним із попередніх значень захисту сторінки.
SEC_COMMIT \ SEC_COMMIT \ SEC_IMAGE_NO_EXECUTE \ SEC_LARGE_PAGES \ SEC_NOCACHE \ SEC_RESERVE \ SEC_WRITECOMBINE
DWORD dwMaximumSizeHigh, //максимального розміру об’єкта відображення файлів
DWORD dwMaximumSizeLow, // Якщо цей параметр і dwMaximumSizeHigh дорівнює 0 (нуль), максимальний розмір об’єкта відображення файлу дорівнює поточному розміру файлу, який ідентифікує hFile
LPCSTR lpName); //
Ім'я об’єкта відображення файлів. Якщо цей параметр відповідає імені існуючого об'єкта відображення, функція вимагає доступу до об'єкта із захистом, який задає

flProtect. Якщо цей параметр NULL, об’єкт відображення файлів створюється без імені. Якщо lpName відповідає імені існуючої події, семафору, mutex, очікуваному таймеру або об'єкту завдання, функція не працює, і функція GetLastError повертає ERROR_INVALID_HANDLE. Це відбувається тому, що ці об'єкти мають однаковий простір імен. Ім'я може мати префікс "Глобальний" або "Локальний", щоб явно створити об'єкт у глобальному просторі або сесії імен. Залишок імені може містити будь-який символ, крім символу зворотної косої риски. Для створення об’єкта відображення файлів у глобальній просторі імен з сеансу, відмінного від нуля сеансу, потрібна привілей SeCreateGlobalPrivilege. Швидке перемикання користувачів здійснюється за допомогою сеансів служби терміналів. Перший користувач для входу використовує сеанс 0 (нуль), наступний користувач для входу використовує сеанс 1 (один) тощо. Імена об’єктів ядра повинні відповідати вказівкам, які викладені для служб терміналів, щоб додатки могли підтримувати декількох користувачів.

Коли ви викликаєте CreateFileMapping, ви вказуєте ім'я об'єкта, кількість байтів, які потрібно відобразити з файлу, та дозвіл на читання / запис для відображеної пам'яті. Перший процес, який викликає CreateFileMapping, створює об'єкт відображення файлів. Процеси, що викликають CreateFileMapping для існуючого об'єкта, отримують обробку для існуючого об'єкта. Ви можете визначити, чи вдалий виклик CreateFileMapping створив або відкрив об’єкт відображення файлів, викликавши функцію GetLastError GetLastError повертає NO_ERROR до процесу створення, а ERROR_ALREADY_EXISTS - наступним процесам.
Функція CreateFileMapping видає помилку, якщо прапори доступу суперечать тим, які вказані, коли функція CreateFile відкрила файл. Наприклад, для читання та запису у файл:

- Вкажіть значення GENERIC_READ та GENERIC_WRITE в параметрі fdwAccess параметра CreateFile.
- Вкажіть значення PAGE_READWRITE в параметрі fdwProtect CreateFileMapping.

Створення об’єкта відображення файлів не займає фізичну пам'ять, воно лише резервує його.
Розмір об'єкта відображення файлу не залежить від розміру файла, який відображається. Однак якщо об’єкт відображення файлу більше, ніж файл, система розгортає файл, перш ніж CreateFileMapping повернеться. Якщо об'єкт відображення файлів менше, ніж файл, система відображає лише вказану кількість байтів з файлу. Параметри dwMaximumSizeHigh та dwMaximumSizeLow у CreateFileMapping дозволяють вказати кількість байтів, які потрібно зіставити з файлу. Якщо ви не хочете, щоб розмір файлу мінявся (наприклад, при зіставленні файлів лише для читання), вкажіть нуль для dwMaximumSizeHigh та dwMaximumSizeLow. При цьому створюється об'єкт відображення файлу, який має точно той же розмір, що і файл. В іншому випадку

потрібно обчислити або оцінити розмір готового файлу, оскільки об’єкти відображення файлів мають статичний розмір; Після створення їх розмір неможливо збільшити чи зменшити. Розмір об'єкта відображення файлів, який підтримується іменованим файлом, обмежений простором на диску. Розмір файлу перегляду обмежений найбільшим доступним суміжним блоком незарезервованої віртуальної пам'яті. Це максимум 2 Гб за мінусом віртуальної пам'яті, яка вже зарезервована процесом.

Щоб відобразити дані з файлу у віртуальну пам'ять процесу, потрібно створити view of the file. Функції MapViewOfFile та MapViewOfFileEx використовують дескриптор об’єкта відображення файлів, повернутих CreateFileMapping, щоб створити view of the file або частини файла у віртуальному просторі адреси процесу. Функція MapViewOfFile повертає вказівник на view of the file. Перенаправляючи вказівник на діапазон адрес, зазначених у MapViewOfFile, програма може читати дані з файлу та записувати дані у файл. Запис у подання до файлу призводить до зміни об’єкта відображення файлів. Фактична запис у файл на диску обробляється системою. Дані фактично не передаються під час запису об'єкта відображення файлів. Натомість значна частина вводу та виводу файлів (вводу / виводу) кешується для покращення загальної продуктивності системи. Програми можуть змінити таку поведінку, викликавши функцію FlushViewOfFile, щоб змусити систему негайно виконувати дискові транзакції.
LPVOID MapViewOfFile(
HANDLE hFileMappingObject,// дескриптор на обєкт відображуваного файлу
DWORD dwDesiredAccess, //тип доступу FILE_MAP_ALL_ACCESS FILE_MAP_READ FILE_MAP_WRITE
DWORD dwFileOffsetHigh,//
DWORD dwFileOffsetLow, //
SIZE_T dwNumberOfBytesToMap);//
Якщо функція успішна, значенням повернення є початкова адреса відображення
Коли процес завершить роботу з об’єктом відображення файлів, він повинен знищити всі перегляди файлів у своєму адресному просторі, використовуючи функцію UnmapViewOfFile для кожного перегляду файлів. Коли кожен процес завершить роботу з об’єктом відображення файлів він повинен закрити дескриптор об'єкта файлу відображення та файл на диску, викликавши CloseHandle. Ці виклики CloseHandle досягають успіху, навіть коли є перегляди файлів, які все ще відкриті. Однак залишення поданих файлів у map режимі спричиняє витік пам'яті.

Приклад взаємодії процесів через використання спільної памяті - файлу підкачки
1-й процес

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

```
#include <stdio.h>
#include <conio.h>
#define BUF_SIZE 256 TCHAR szName[]=TEXT("MyFileMappingObject");
TCHAR szMsg[]=TEXT("Message from first process");
void main()
{  HANDLE hMapFile;
  LPCTSTR pBuf;

  hMapFile = CreateFileMapping(
            INVALID_HANDLE_VALUE,    // использование файла подкачки
       NULL,              // защита по умолчанию
       PAGE_READWRITE,       // доступ к чтению/записи
       0,                 // макс. размер объекта
       BUF_SIZE,             // размер буфера
       szName);              // имя отраженного в памяти объекта

  if (hMapFile == NULL || hMapFile == INVALID_HANDLE_VALUE)
  {
    printf("Не может создать  отраженный в памяти объект (%d).\n",
    GetLastError());
    return;
  }   pBuf = (LPTSTR) MapViewOfFile(
                        hMapFile,       // дескриптор отраженного // в памяти объекта
       FILE_MAP_ALL_ACCESS, // разрешение чтения/записи
                  0,
                  0,
                  BUF_SIZE);

  if (pBuf == NULL)
  {
        printf("Представление проецированного файла не возможно (%d).\n",
    GetLastError());
    return;
  }

  CopyMemory((PVOID)pBuf, szMsg, strlen(szMsg));
  getch();

  UnmapViewOfFile(pBuf);

  CloseHandle(hMapFile);
}
```

Другий процес може одержати доступ до тих самих даних

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#define BUF_SIZE 256TCHAR
 szName[]=TEXT("MyFileMappingObject");
void main()
{  HANDLE hMapFile;
  LPCTSTR pBuf;

  hMapFile = OpenFileMapping(
              FILE_MAP_ALL_ACCESS,   // доступ к чтению/записи
              FALSE,                 // имя не наследуется
              szName);               // имя "проецируемого " объекта

  if (hMapFile == NULL)
  {
    printf("Невозможно открыть объект "проекция файла" (%d).\n",
    GetLastError());
    return;
  }

   pBuf = MapViewOfFile(hMapFile,     // дескриптор "проецируемого"
объекта
            FILE_MAP_ALL_ACCESS,  // разрешение чтения/записи
            0,
            0,
            BUF_SIZE);

  if (pBuf == NULL)
  {
          printf("Представление проецированного файла не возможно
(%d).\n",
    GetLastError());
    return;
  }
  MessageBox(NULL, pBuf, TEXT("Process2"), MB_OK);

  UnmapViewOfFile(pBuf);

  CloseHandle(hMapFile);
}
```

Приклад з використанням синхронізації

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

```
HANDLE hEventChar;
HANDLE hEventTermination;
HANDLE hEvents[2];
CHAR lpEventName[] =  "$MyVerySpecialEventName$";
CHAR lpEventTerminationName[] =   "$MyVerySpecialEventTerminationName$";
CHAR lpFileShareName[] =   "$MyVerySpecialFileShareName$";
HANDLE hFileMapping;
LPVOID lpFileMap;
int main()
{
 DWORD dwRetCode;
 printf("Mapped and shared file, server process\n");
 hEventChar = CreateEvent(NULL, FALSE, FALSE, lpEventName);
  if(hEventChar == NULL)
 {
  fprintf(stdout,"CreateEvent: Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }
 if(GetLastError() == ERROR_ALREADY_EXISTS)
 {
  printf("\nApplication EVENT already started\n"
    "Press any key to exit...");
  getch();
  return 0;
 }

 hEventTermination = CreateEvent(NULL,
  FALSE, FALSE, lpEventTerminationName);
 if(hEventTermination == NULL)
 {
  fprintf(stdout,"CreateEvent (Termination): Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }
 hFileMapping = CreateFileMapping((HANDLE)0xFFFFFFFF,
  NULL, PAGE_READWRITE, 0, 100, lpFileShareName);
 if(hFileMapping == NULL)
 {
```

```
    fprintf(stdout,"CreateFileMapping: Error %ld\n",
      GetLastError());
    getch();
    return 0;
  }
  lpFileMap = MapViewOfFile(hFileMapping,
    FILE_MAP_READ | FILE_MAP_WRITE, 0, 0, 0);
  if(lpFileMap == 0)
  {
    fprintf(stdout,"MapViewOfFile: Error %ld\n",
      GetLastError());
    getch();
    return 0;
  }
   hEvents[0] = hEventTermination;
  hEvents[1] = hEventChar;
  while(TRUE)
  {
      dwRetCode = WaitForMultipleObjects(2,
     hEvents, FALSE, INFINITE);
    if(dwRetCode == WAIT_ABANDONED_0     ||
       dwRetCode == WAIT_ABANDONED_0 + 1 ||
       dwRetCode == WAIT_OBJECT_0        ||
       dwRetCode == WAIT_FAILED)
      break;
    putch(*((LPSTR)lpFileMap));
  }
  CloseHandle(hEventChar);
  CloseHandle(hEventTermination);
  UnmapViewOfFile(lpFileMap);
  CloseHandle(hFileMapping);
  return 0;
}

#include <windows.h>
#include <stdio.h>
#include <conio.h>
HANDLE hEvent;
HANDLE hEventTermination;
CHAR lpEventName[] =   "$MyVerySpecialEventName$";
CHAR lpEventTerminationName[] =   "$MyVerySpecialEventTerminationName$";
```

```
CHAR lpFileShareName[] =  "$MyVerySpecialFileShareName$";
HANDLE hFileMapping;
LPVOID lpFileMap;

int main()
{
 CHAR chr;

 printf("Mapped and shared file, client process\n"
      "\n\nPress <ESC> to terminate...\n");
 hEvent = OpenEvent(EVENT_ALL_ACCESS, FALSE, lpEventName);

 if(hEvent == NULL)
 {
  fprintf(stdout,"OpenEvent: Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }

  hEventTermination = OpenEvent(EVENT_ALL_ACCESS,
  FALSE, lpEventTerminationName);

 if(hEventTermination == NULL)
 {
  fprintf(stdout,"OpenEvent (Termination): Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }
 hFileMapping = OpenFileMapping(
  FILE_MAP_READ | FILE_MAP_WRITE, FALSE, lpFileShareName);
 if(hFileMapping == NULL)
 {
  fprintf(stdout,"OpenFileMapping: Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }
 lpFileMap = MapViewOfFile(hFileMapping,
  FILE_MAP_READ | FILE_MAP_WRITE, 0, 0, 0);
```

```
 if(lpFileMap == 0)
 {
  fprintf(stdout,"MapViewOfFile: Error %ld\n",
    GetLastError());
  getch();
  return 0;
 }
 while(TRUE)
 {
  chr = getche();
    if(chr == 27)
    break;
  *((LPSTR)lpFileMap) = chr;
  SetEvent(hEvent);
 }
  SetEvent(hEvent);
 SetEvent(hEventTermination);
  CloseHandle(hEvent);
 CloseHandle(hEventTermination);
 UnmapViewOfFile(lpFileMap);
 CloseHandle(hFileMapping);
 return 0;
}
```

Завдання.

1. Реалізувати алгоритм моделювання заданої задачі за допомогою окремих процесів згідно індивідуального завдання.

2. Реалізувати синхронізацію роботи процесів.

3. Забезпечити зберігання результатів виконання завдання.

4. Результати виконання роботи відобразити у звіті.

Індивідуальні завдання.

Варіант - 1 Реалізувати міжпроцесну взаємодію одним із відомих вам методів. Один із процесів має бути сервером, який дозволяє процесам-клієнтам підписатись/відписатись на один із сервісів розсилки (щогодинний прогноз погоди, щохвилинний курс акцій, щоденний курс валют). Для збереження інформації на сервері можна використати бази даних. Реалізувати дану модель, використовуючи, що проектуються у пам’ять або пайпи (робота в межах однієї системи).

Варіант - 2 Реалізувати міжпроцесну взаємодію одним із відомих вам методів. Один із процесів має бути сервером, який здійснює моніторинг за файлами в деякій директорії, повертає інформацію про них (сумарний розмір, список файлів, та дату створення файлів). Процеси-клієнти надсилають запит на сервер, в якому вказують розширення файлів, що їх цікавлять і директорій, в який буде здійснюватись пошук. Можна вважати, що файли не змінюються жодним чином впродовж 5 секунд, тому для для запитів, які приходять в інтервалі 5 секунд використати кешування.

Варіант-3 Створити програму, що моделює наступну ситуацію: Процес-науковий керівник проекту пропонує виконавців проекту-дочірні процеси. Процес-керівник створює додаток-віртуальну дошку (файл), де можна генерувати ідеї для проекту. Процеси-виконавці генерують ідеї, записуючи їх на спільну дошку. На виконання даного завдання вони мають 3 хвилини, після чого процес-керівник призупиняє їхню роботу і виводить на екран усі згенеровані ідеї, нумеруючи кожну з них. Процеси-виконавці голосують за три найкращі ідеї. Після чого процес-керівник записує на дошку три найкращі ідеї і закриває роботу додатку-віртуальної дошки, зберігаючи її вміст. Реалізувати дану модель, використовуючи а) файли, що проектуються у пам’ять; пайпи (робота в межах однієї системи)

Варіант -4. Створити програму, що моделює наступну ситуацію: Модератори форуму. Користувач реєструється на форумі і його ім’я записується у базу-даних (файл). Після того він може написати повідомлення. Викликаються модератори форуму. Кожен з яких слідкує за певним забороненим словом. Повідомлення може бути виведене на екран, якщо загальна сума заборонених слів не перевищує певне число. Кількість заборонених слів заноситься в базу даних. На форумі можуть працювати кілька користувачів.

Варіант -5. Створити програму-чат спілкування між процесами. Процес, що запускає чат, надає можливість вибору варіанту методу спілкування з 2-3 методів міжпроцесної взаємодії.

Варіант-6 (основа як варіант 3), реалізувати, використовуючи б) сокети

Варіант - 7 Взяти за основу завдання із варіанту 1. Реалізувати, використовуючи сокети.