

Университет ИТМО

**Лабораторная работа №3
по дисциплине «Операционные системы»**

Выполнил:
студент 3-го курса
группы 3125
Припадчев Артём

Санкт-Петербург
2014

Вариант 8

Задание: Написать программы для консольного процесса **Administrator** и консольных процессов **Reader** и **Writer**.

Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение "A", сообщение "B", и конец сеанса для процессов **Reader** и **Writer**.

Одновременно принимать и отправлять сообщения могут **только один процесс Writer** и **один процесс Reader**, передача остальных сообщений от других процессов должна блокироваться с помощью мьютексов;

Процесс **Administrator**:

- запрашивает у пользователя количество процессов **Reader** и **Writer**, которые он должен запустить;
- запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer** и кол-во принятых сообщений для процесса **Reader**(соответствие сообщений проверить и подкорректировать по формуле);
- запускает заданное количество процессов **Reader** и **Writer**;
- принимает от каждого процесса **Reader** и **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
- завершает свою работу.

Процесс **Writer**:

- запрашивает с консоли сообщения, и передает их (по одному) процессу **Reader**; передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- принимает сообщение от процесса **Writer**; выводит на консоль сообщение;
- передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу.

Исходные коды

Administrator

```
...
hMutexR1 = CreateMutex(NULL, FALSE, L"MutexR1");
hMutexW1 = CreateMutex(NULL, FALSE, L"MutexW1");
hMsgA = CreateEvent(NULL, TRUE, FALSE, L"MessageA");
hEndW = CreateEvent(NULL, TRUE, FALSE, L"EndW");
hEndR = CreateEvent(NULL, TRUE, FALSE, L"EndR");
StringCbPrintf(buf, MAX_BUF, L"%d", msgCount);
SetEnvironmentVariable(L"msgCount", buf);
int i = 0;
while (i < pCount)
{
    i++;
    STARTUPINFO si, si2;
    PROCESS_INFORMATION pi, pi2;
    ZeroMemory(&si, sizeof(STARTUPINFO));
    si.cb = sizeof(STARTUPINFO);
    ZeroMemory(&si2, sizeof(STARTUPINFO));
    si2.cb = sizeof(STARTUPINFO);

    if (!CreateProcess(L"Writer.exe", NULL, NULL, NULL, FALSE, CREATE_NEW_CONSOLE,
NULL, NULL, &si, &pi))
    {
        printf("Can't start Writer.exe\r\n");
        return GetLastError();
    }
}
```

```

        if (!CreateProcess(L"Reader.exe", NULL, NULL, NULL, FALSE, CREATE_NEW_CONSOLE,
NULL, NULL, &si2, &pi2))
        {
            printf("Can't start Reader.exe\r\n");
            return GetLastError();
        }
    }

    int ends = 0;
    HANDLE hEnd[2];
    hEnd[0] = OpenEvent(EVENT_ALL_ACCESS, FALSE, L"EndW");
    hEnd[1] = OpenEvent(EVENT_ALL_ACCESS, FALSE, L"EndR");

    while (ends < pCount)
    {
        WaitForMultipleObjects(2, hEnd, TRUE, INFINITE);
        ResetEvent(hEnd[0]);
        ResetEvent(hEnd[1]);
        ends++;
        printf("Writer and Reader session closed\r\n");
    }

    printf("\r\nInput any char to exit...\r\n");
    scanf_s("%d", &pCount);
    CloseHandle(hMutexR1);
    CloseHandle(hMutexW1);
    CloseHandle(hMsgA);
    CloseHandle(hEndW);
    CloseHandle(hEndR);
    return 0;

```

...

Writer

...

```

HANDLE hMutex[1];
HANDLE hMsg, hEnd;
hMutex[0] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexW1");
if (hMutex[0] == NULL)
    printf("Can't open Mutex\r\n");
FILE *f;
char fName[16];
printf("Waiting for ending other Writers...\r\n");
while (true) // Îæèääâï äðóäèâ ïðïöãñû writer
{
    if (WaitForSingleObject(hMutex[0], 500) == WAIT_OBJECT_0)
    {
        hMsg = OpenEvent(EVENT_MODIFY_STATE, FALSE, L"MessageA");
        strcpy(fName, "MessageA.txt");
        break;
    }
}
hEnd = OpenEvent(EVENT_MODIFY_STATE, FALSE, L"EndW");
if (hMsg == NULL || hEnd == NULL)
    printf("Can't open Event\r\n");
int i = 0;
TCHAR buf[MAX_BUF] = {0};
GetEnvironmentVariable(L"msgCount", buf, MAX_BUF);
LPTSTR endPtr;
int msgCount = _tcstod(buf, &endPtr);
while (i < msgCount)
{

```

```

    printf("Enter %d message(-s):\r\n", msgCount - i);
    _tscanf(L"%s", buf, 255);
    f = fopen(fName, "wt");
    fwrite(buf, sizeof(TCHAR), _tcslen(buf), f);
    fclose(f);
    printf("Sending message...\r\n");
    SetEvent(hMsg);
    Sleep(1000);
    i++;
}
SetEvent(hEnd);
ReleaseMutex(hMutex[0]);
CloseHandle(hMutex[0]);
CloseHandle(hMsg);
CloseHandle(hEnd);
return 0;

```

...

Reader

...

```

HANDLE hMutex[1];
HANDLE hMsg, hEnd;
hMutex[0] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexR1");
if (hMutex[0] == NULL)
    printf("Can't open Mutex\r\n");
FILE *f;
char fName[16];
printf("Waiting for ending other Readers...\r\n");
while (true)
{
    if (WaitForSingleObject(hMutex[0], 500) == WAIT_OBJECT_0)
    {
        hMsg = OpenEvent(EVENT_ALL_ACCESS, FALSE, L"MessageA");
        strcpy(fName, "MessageA.txt");
        break;
    }
}
hEnd = OpenEvent(EVENT_MODIFY_STATE, FALSE, L"EndR");
if (hMsg == NULL || hEnd == NULL)
    printf("Can't open Event\r\n");
int i = 0;
TCHAR buf[MAX_BUF] = {0};
GetEnvironmentVariable(L"msgCount", buf, MAX_BUF);
LPTSTR endPtr;
int msgCount = _tcstod(buf, &endPtr);
while (i < msgCount)
{
    printf("Waiting message...\r\n");
    DWORD dwRes = WaitForSingleObject(hMsg, INFINITE);
    if (dwRes != WAIT_OBJECT_0)
        printf("Wait for single object failed\r\nERROR: %d ERRORCODE: %d\r\n",
dwRes, GetLastError());
    f = fopen(fName, "rt");
    memset(buf, 0, sizeof(buf));
    fread(buf, sizeof(TCHAR), MAX_BUF, f);
    fclose(f);
    _tprintf(L"Message recieved: %s\r\n", buf);
    ResetEvent(hMsg);
    i++;
}

```

```
SetEvent(hEnd);  
printf("\r\nClose in 5 seconds...\r\n");  
Sleep(5000);  
ReleaseMutex(hMutex[0]);  
CloseHandle(hMutex[0]);  
CloseHandle(hMsg);  
CloseHandle(hEnd);  
return 0;
```

...

Вывод: в процессе выполнения работы были изучены подходы к управлению разделяемыми ресурсами. Для этой цели использовались мьютексы, события и соответствующие им методы ожидания для процессов.