Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Операционные системы

Лабораторная работа 4

"Синхронизация процессов при помощи семафор и мьютексов"

Вариант 9

Выполнил: Шкаруба Н.Е.

Группа: P3318

2016 г

Задание

Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение “A” , сообщение “B”, и конец сеанса для процессовReaderиWriter.

Одновременно принимать и отправлять сообщения могут только два процессаWriter и два процессаReader, передача остальных сообщений от других процессов должна блокироваться с помощью мьютексов;

Процесс Administrator:

- запрашивает у пользователя количество процессов ReaderиWriter, которые он должен запустить;
- запрашивает у пользователя кол-во отправленных сообщений для процесса Writer.
- Кол-во принятых сообщений для процесса Reader вычислить. (соответствие сообщений проверить иподкорректировать по формуле);
- запускает заданное количество процессов ReaderиWriter;
- принимает от каждого процесса ReaderиWriter 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.

Код Администратора:

```
int main() {
	if (!getCountReadersAndWriters() || !getCountMessages()) {
		printf("Incorrect value!\r\n");
		return 0;
	}

	hMutexR1 = CreateMutex(NULL, FALSE, L"MutexR1");
	hMutexR2 = CreateMutex(NULL, FALSE, L"MutexR2");
	hMutexW1 = CreateMutex(NULL, FALSE, L"MutexW1");
	hMutexW2 = CreateMutex(NULL, FALSE, L"MutexW2");

	hMsgA = CreateEvent(NULL, TRUE, FALSE, L"MessageA");
	hMsgB = CreateEvent(NULL, TRUE, FALSE, L"MessageB");

	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");
			system("pause");
			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");
			system("pause");
			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(hMutexR2);
	CloseHandle(hMutexW1);
	CloseHandle(hMutexW2);

	CloseHandle(hMsgA);
	CloseHandle(hMsgB);
	CloseHandle(hEndW);
	CloseHandle(hEndR);

	return 0;
}
```

Код Writer'a:

```
int main() {
	HANDLE hMutex[2];
	HANDLE hMsg, hEnd;

	hMutex[0] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexW1");
	hMutex[1] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexW2");

	if (hMutex[0] == NULL || hMutex[1] == NULL)
		printf("Can't open Mutex\r\n");

	FILE *f;
	char fName[16];
	FILE *stream;
	errno_t err;

	printf("Waiting for ending other Writers...\r\n");
	while (true) {
		if (WaitForSingleObject(hMutex[0], 500) == WAIT_OBJECT_0) {
			hMsg = OpenEvent(EVENT_MODIFY_STATE, FALSE, L"MessageA");
			strcpy_s(fName, "MessageA.txt");
			break;
		}
		if (WaitForSingleObject(hMutex[1], 500) == WAIT_OBJECT_0) {
			hMsg = OpenEvent(EVENT_MODIFY_STATE, FALSE, L"MessageB");
			strcpy_s(fName, "MessageB.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);
		wscanf_s(L"%s", buf, 255);

		err = fopen_s(&stream, fName, "wt");
		fwrite(buf, sizeof(TCHAR), _tcslen(buf), stream);
		fclose(stream);

		printf("Sending message...\r\n");
		SetEvent(hMsg);
		Sleep(1000);
		i++;
	}

	SetEvent(hEnd);
	ReleaseMutex(hMutex[0]);
	ReleaseMutex(hMutex[1]);
	CloseHandle(hMutex[0]);
	CloseHandle(hMutex[1]);
	CloseHandle(hMsg);
	CloseHandle(hEnd);

	return 0;
}
```

Код Reader'a:

```
int main() {
	HANDLE hMutex[2];
	HANDLE hMsg, hEnd;

	hMutex[0] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexR1");
	hMutex[1] = OpenMutex(SYNCHRONIZE, FALSE, L"MutexR2");

	if (hMutex[0] == NULL || hMutex[1] == NULL)
		printf("Can't open Mutex\r\n");

	FILE *f;
	char fName[16];
	FILE *stream;
	errno_t err;

	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_s(fName, "MessageA.txt");
			break;
		}
		if (WaitForSingleObject(hMutex[1], 500) == WAIT_OBJECT_0) {
			hMsg = OpenEvent(EVENT_ALL_ACCESS, FALSE, L"MessageB");
			strcpy_s(fName, "MessageB.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());

		err = fopen_s(&stream, fName, "rt");
		memset(buf, 0, sizeof(buf));
		fread(buf, sizeof(TCHAR), MAX_BUF, stream);
		fclose(stream);

		_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]);
```

```
        ReleaseMutex(hMutex[1]);
        CloseHandle(hMutex[0]);
        CloseHandle(hMutex[1]);
        CloseHandle(hMsg);
        CloseHandle(hEnd);

        return 0;
}
```