Základní práce s vlákny ve Windows

27. dubna 2022

V operačním systému Microsoft Windows základní jednotkou vykonávající program[1] je *vlákno*. V každém procesu může běžet jedno nebo více vláken, kdy vlákno může vykonávat libovolnou část kódu programu, včetně částí, které vykonávají jiná vlákna. Při spuštění procesu je spuštěno (primární) vlákno, které vykonává funkci `main`. Každé vlákno má vlastní kontext, tj. má přidělené registry a zásobník. Další prostředky zejména paměť[2], systémové prostředky[3] jsou sdíleny mezi všemi vlákny, což umožňuje pohodlnou spolupráci více vláken při řešení úloh. Avšak je potřeba mít na paměti, že běh jednotlivých vláken je plánován operačním systémem a činnosti, jež vlákna provádí, se mohou téměř libovolně prolínat. Je proto nutné zajistit, aby v jeden okamžik s jedněmi daty nepracovalo více vláken současně. Toho lze dosáhnout buď pomocí synchronizace vláken nebo vhodně navrženým kódem, který zajistí, že nebudou měněna data, se kterými pracuje více vláken současně.

1 Vytvoření vlákna

K vytvoření vlákna slouží funkce `CreateThread`[4], které vedle atributů definujících vlastnosti vytvořeného vlákna předáváme především odkaz na funkci, která představuje kód vykonáváný vláknem a odkaz na data, se kterými vlákno pracuje. Vytvoření vlákna ilustruje následující kód.

```
1  #include <windows.h>
2  #include <tchar.h>
3  #include <stdio.h>
4
5  #define COUNT   (20)
6
7  DWORD WINAPI ThreadFunc(LPVOID lpParam)
8  {
9          _tprintf(_T("Spusteno vlakno s id: %i\n"), GetCurrentThreadId());
10         DWORD x = *(DWORD *)lpParam;
11         for (int i = 0; i < COUNT; i++) {
12                 _tprintf(_T("thr #%i: %i\n"), x, i);
```

[1] instrukce programu
[2] včetně prováděného kódu
[3] objekty poskytované operačním systémem
[4] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread`

```
13                  Sleep(1);
14          }
15
16          return 0;
17  }
18
19  int _tmain()
20  {
21          DWORD dwThreadId;
22          DWORD dwThrdParam = 1;
23
24          HANDLE hThread = CreateThread(
25                  NULL,                           // bezpecnostni atributy
26                  0,                              // velikost zasobniku (0 -> implicitni hodnota)
27                  ThreadFunc,                     // funkce provadena vlakna
28                  &dwThrdParam,                   // argument predany vlaknu
29                  0,                              // priznaky pro vytvorene vlakno
30                  &dwThreadId);                   // vraci id vlakna
31
32          if (hThread == NULL) {
33                  _tprintf(_T("Vytvoreni vlakna selhalo\n"));
34                  ExitProcess(0);
35          } else {
36                  _tprintf(_T("Vytvoreno vlakno s id: %i\n"), dwThreadId);
37          }
38
39          for (int i = 0; i < COUNT; i++) {
40                  _tprintf(_T("thr #0: %i\n"), i);
41                  Sleep(1);
42          }
43
44          WaitForSingleObject(hThread, INFINITE);  // ceka na skonceni vlakna
45          CloseHandle(hThread);                    // ukonci praci s vlaknem
46  }
```

Při popisu tohoto kódu začneme uprostřed, na řádcích 24 až 30, kde dochází k vytvoření vlákna. Důležitý je třetí argument na řádku 27 udávající funkci, kterou bude vlákno vykonávat. Velmi často používáný je i čtvrtý argument na řádku 28, kterým formou ukazatele můžeme do vlákna předat data, se kterými se bude pracovat.[5] Funkce `CreateThread` vrací handle na dané vlákno, se kterým můžeme následně pracovat. Zejména bychom měli (i) otestovat, zda došlo k vytvoření vlákna (řádky 32 až 35), (ii) počkat na

[5]Pokud předáváme odkaz na lokální proměnné, tj. data, která jsou alokována na zásobníku, je nutné zajistit, aby vlákno jež takto data předává neskončilo dřív než vlákno jím spuštěné. Jinak by společně se skončeným vláknem zanikla i data umístěná na jeho zásobníku. Alternativně můžeme předávat dynamicky alokovaná data.

dokončení vlákna (řádek 44) a ukončit práci s daným vláknem (řádek 45).

Součástí ukázkového kódu je i zjištění identifikátoru vlákna, viz šestý argument funkce `CreateThread` a řádek 36. Abychom vyvíjeli nějakou činnost v primárním vláknu, obsahuje funkce `main` na řádcích smyčku, která vypisuje čísla od 0 do `COUNT` a po každém výpisu se na 1 ms uspí pomocí volání funkce `Sleep`[6].

Funkce vykonávající vlákno je typu `DWORD WINAPI ThreadFunc(LPVOID lpParam)`, kde argument `lpParam` je ukazatel předaný funkci `CreateThread`. V těle funkce můžeme vykonávat libovolný kód, jak ukazuje náš příklad na řádcích 7 až 17. V této funkci nejdříve vypíšeme id vlákna. Při spuštění programu si všimněme, že moment vytvoření vlákna a jeho spuštění nemusí být stejný. Ve výpisu programu se to projeví tak, že primární vlákno vstoupí do smyčky na řádcích 39 až 42 a až následně dojde ke spuštění vlákna.

Dále vlákno převezme svůj argument a začne ve smyčce (řádek 11 až 14) vypisovat text. Funkce vrací hodnotu typu `DWORD`, kterou můžeme předat informaci související s ukončením vlákna. Tuto hodnotu můžeme vyzvednout pomocí funkce `GetExitCodeThread`[7].

Úkoly:

1. Odstraňte z kódu volání `Sleep`, případně změnte jeho argumenty, a sledujte, jak se změní průběh programu.
2. Rozšiřte ukázkový program tak, aby vyzvedl a vypsal návratovou hodnotu spuštěného vlákna. Zamyslete se nad tím, kam volání funkce `GetExitCodeThread` umístit.
3. Rozšiřte ukázkový program, aby pracoval obecně s `N` vlákny, kde `N` je konstanta zadaná v kódu programu.

2 Manipulace s vlákny

Operační systém Microsoft Windows nabízí širokou škálu funkcí pro práci s vlákny, některé již byly zmíněny v předchozí kapitole. Zmiňme si dále funkce:

- `GetCurrentThread`[8] vracející pseudo-handle sloužící k manipulaci s aktuálním vláknem. Pseudo-handle zde funguje jako zvláštní konstanta odkazující na aktuální vlákno a má platnost pouze v rámci daného vlákna. Pokud bychom chtěli získat plnohodnotný handle, musíme pseudo-handle převést pomocí funkce `DuplicateHandle`[9].
- `SuspendThread`[10] a `ResumeThread`[11] sloužící k uspání a probuzení vlákna. Funkce `SuspendThread` může být opakovaně zavolána i na již uspané vlákno. Každé vlákno má přiřazeno počítadlo, které indikuje počet uspání daného vlákna. Funkce `ResumeThread` toto počítadlo snižuje, a pokud toto počítadlo klesne na 0, je vlákno probuzeno.

[6] https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep
[7] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getexitcodethread
[8] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getcurrentthread`
[9] `https://docs.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-duplicatehandle`
[10] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-suspendthread`
[11] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-resumethread`

- `ExitThread`[12] ukončí právě prováděné vlákno.
- `TerminateThread`[13] ukončí jiné vlákno na základě předaného handlu. Ukončení vlákna touto funkcí není korektní, může k němu dojít v libovolném bodě daného vlákna a tím pádem se může program dostat do nekonzistentního stavu.

Úkoly:

4. Ukázkový příklad upravte tak, že po provedení `(COUNT / 2)` cyklů se vlákno uspí a je probuzeno v momentě, kdy primární vlákno zpracuje celou smyčku.
5. Naprogramujte funkci `int parfib(int)`, která spočítá Fibonacci číslo rekurzivní způsobem s využitím právě dvou vláken. Dvě počáteční větve výpočtu spusťte v samostatných vláknech.
6. Naprogramujte funkci `int pmin(int *numbers, unsigned int count, unsigned int threads)`, která použije `threads` vláken k tomu, aby v poli `numbers`, které obsahuje `count` hodnot, našlo nejmenší hodnotu.

[12] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-exitthread`
[13] `https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminatethread`