Thread-uri - Extra

Linux

Terminarea thread-urilor

Există şi posibilitatea ca un fir de execuţie să termine un alt fir, folosind mecanismul de "*thread cancellation*". Pentru a face acest lucru se foloseşte funcţia `pthread_cancel` care primeşte ca parametru id-ul firului de execuţie ce urmează să fie terminat :

```
int pthread_cancel(pthread_t thread);
```

Un thread unificabil care a fost terminat în acest mod trebuie aşteptat folosind `pthread_join` pentru ca resursele folosite de el să fie eliberate. Un fir terminat cu `pthread_cancel` va întoarce valoarea PTHREAD_CANCELED.
Totuşi trebuie avut grijă la folosirea acestui mecanism, întrucât un thread este posibil să fie terminat înainte de a avea posibilitatea să elibereze anumite resurse folosite. Din aceasta cauză un thread poate să controleze dacă şi când poate fi terminat de către un alt thread.
Din punctul de vedere al posibilităţii terminării folosind `pthread_cancel` un thread poate fi :

- cancelabil asincron - un astfel de fir poate fi terminat de către un alt fir în orice punct al execuţiei.
- cancelabil sincron - un astfel de fir NU poate fi terminat în orice punct al execuţiei sale, ci numai în anumite puncte specifice.
- necancelabil - un astfel de fir nu poate fi terminat folosind `pthread_cancel`.

Stabilirea tipului unui fir de execuţie din acest punct de vedere se face folosind funcţiile :

```
int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
```

Funcţia `pthread_setcancelstate` poate fi folosită pentru a activa/dezactiva posibilitatea terminării unui fir folosind `pthread_cancel`. Argumentul `state` reprezintă noua lui stare care poate fi `PTHREAD_CANCEL_ENABLE` (pentru activare) sau `PTHREAD_CANCEL_DISABLE` (pentru dezactivare). În `oldstate`, dacă nu este NULL, se poate obţine vechea stare.

Folosind `pthread_setcancelstate` pot fi implementate regiuni critice, în sensul că tot codul dintr-o astfel de regiune trebuie executat în întregime sau deloc, practic firul să nu poată fi terminat de către un alt fir în timp ce se găseşte într-o astfel de regiune.

Funcţia `pthread_setcanceltype` poate fi folosită pentru a schimba tipul de răspuns la o cerere de terminare : asincron sau sincron. Argumentul `type` indică noul tip şi poate fi `PTHREAD_CANCEL_ASYNCHRONOUS` (pentru ca firul să poată fi terminat asincron, deci oricând) sau `PTHREAD_CANCEL_DEFERRED` (pentru ca o cerere de terminare sa fie amânată până când se ajunge într-un punct în care este posibilă terminarea firului). La fel, în `oldtype`, dacă nu este NULL se poate obţine starea anterioară.

În mod implicit la crearea unui fir folosind `pthread_create` starea lui este caracterizată de `PTHREAD_CANCEL_ENABLE` şi `PTHREAD_CANCEL_DEFERRED`.

Funcţiile `pthread_create`, `pthread_setcancelstate` şi `pthread_setcanceltype` întorc 0 în caz de succes şi un cod de eroare nenul în caz de eşec.

În cazul în care un fir este cancelabil sincron, aşa cum a fost precizat, terminarea lui se poate face numai în anumite puncte ale execuţiei sale. Practic cererile de terminare sunt amânate până se ajunge într-un astfel de punct, numit şi `cancellation point`. Când se ajunge într-un astfel de punct se testează dacă există cereri de terminare, şi dacă da, firul este terminat. Cel mai direct mod de a crea un astfel de punct este apelând funcţia :

```
void pthread_testcancel(void);
```

Dacă într-un program se foloseşte acest mecanism de terminare a firelor folosind `pthread_cancel`, această funcţie ar trebui apelată periodic în cadrul unei funcţii asociate unui thread în care se fac multe procesări, în punctele în care firul poate fi terminat fără a rămâne resurse neeliberate şi fără a produce alte efecte nedorite.

Pe lângă `pthread_testcancel` mai există o serie de alte funcţii al căror apel reprezintă un punct de cancelare. Aceste funcţii sunt următoarele :

- pthread_join
- pthread_cond_wait
- pthread_cond_timedwait
- sem_wait
- sigwait

De asemenea, orice funcţie care apelează una din aceste funcţii este şi ea un astfel de punct de terminare.

În general nu este o idee foarte bună folosirea funcţiei pthread_cancel pentru a termina un thread, decât în cazuri excepţionale. În cazuri normale o strategie mai bună este de a indica firului că trebuie să se termine şi apoi să se aştepte terminarea lui.

Lucrul cu atributele unui thread

Atributele unui fir de execuţie (cu o excepţie) sunt specificate la crearea firului de execuţie şi *nu pot fi schimbate* pe perioada în care firul de execuţie este folosit.

Pentru *iniţializarea* şi respectiv *distrugerea* unui obiect ce reprezintă atributele unui fir de execuţie avem la dispoziţie funcţiile :

```
int pthread_attr_init(pthread_attr_t *tattr);
int pthread_attr_destroy(pthread_attr_t *tattr);
```

Pentru a stabili anumite atribute specifice ale unui fir, trebuie urmaţi câţiva paşi :

1. se creează un obiect de tipul pthread_attr_t, de exemplu declarând o variabilă de acest tip.
2. se apelează funcţia pthread_attr_init căreia i se dă ca parametru un pointer la acest obiect. Această funcţie iniţializează atributele cu valorile lor implicite.
3. se modifică obiectul ce contine atributele folosind una din funcţiile prezentate mai jos, pentru ca să se obţină atributele dorite.
4. se transmite un pointer la aceste atribute funcţiei pthread_create.
5. se apelează funcţia pthread_attr_destroy pentru a elibera obiectul ce reprezintă atributele (variabila de tip pthread_attr_t nu este însă dezalocată, ea poate fi refolosită utilizând pthread_attr_init).

Un acelaşi obiect de tip pthread_attr_t poate fi folosit pentru crearea mai multor fire de execuţie distincte şi nu este necesar să fie păstrat după crearea acestora.

În continuare sunt prezentate funcţiile ce modifică atributele cele mai uzuale ale unui fir de execuţie.

Modificarea atributului detaşabil/unificabil

Pentru a seta/verifica tipul unui fir de execuţie din punct de vedere detaşabil/unificabil se pot utiliza următoarele funcţii :

```
int pthread_attr_setdetachstate(pthread_attr_t *tattr, int detachstate);
int pthread_attr_getdetachstate(const pthread_attr_t *tattr, int *detachstate);
```

Primul parametru este un pointer la obiectul reprezentând atributele, iar al doilea parametru reprezintă starea dorită - unificabil/detaşabil. Deoarece implicit sunt create fire unificabile (valoarea PTHREAD_CREATE_JOINABLE), această funcţie trebuie apelată doar dacă se creează fire detaşabile şi în acest caz al doilea parametru trebuie să aibă valoarea PTHREAD_CREATE_DETACHED.

Chiar dacă un fir de execuţie este creat unificabil, el poate fi transformat ulterior într-un fir detaşabil apelând funcţia pthread_detach. Însă o data detaşat, nu mai poate fi făcut unificabil la loc.

Modificarea politicii de alocare a procesorului

Standardul POSIX defineşte 3 politici de alocare a procesorului :

- SCHED_RR - round robin
- SCHED_FIFO - first in first out
- SCHED_OTHER - implementare dependentă de sistem

Politicile SCHED_RR şi SCHED_FIFO sunt opţionale şi sunt suportate DOAR de firele de execuţie real time.

Funcţia care este folosită pentru a schimba politica de execuţie a firelor este :

```
int pthread_attr_setschedpolicy(pthread_attr_t *tattr, int policy);
```

Pentru a afla politica curentă se poate folosi funcţia `pthread_attr_getschedpolicy` care în momentul de faţă întoarce doar SCHED_OTHER.

```
int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
```

Modificarea priorităţii

Pentru a schimba/verifica prioritatea firelor de execuţie dispunem de următoarele funcţii :

```
int pthread_attr_setschedparam(pthread_attr_t *tattr, const struct sched_param *param);
int pthread_attr_getschedparam(pthread_attr_t *tattr, struct sched_param *param);
```

Prioritatea este semnificativă doar dacă politica folosită este SCHED_RR sau SCHED_FIFO.

Modificarea dimensiunii şi adresei de start a stivei

De obicei stivele firelor de execuţie încep la limita unei pagini de memorie, iar orice dimensiune a lor este rotunjită până la limita următoarei pagini. La capătul stivei este adăugată de obicei o pagină pentru care nu avem drepturi de acces şi astfel cele mai multe depăşiri de stivă (stack overflows) vor genera semnalul SIGSEGV (deci un segmentation fault).

Dacă firul a fost creat unificabil stiva nu poate fi eliberată până nu se va termina un apel `pthread_join` pentru respectivul fir.

De obicei biblioteca de fire de execuţie alocă 1M de memorie virtuală pentru fiecare stivă de fir de execuţie.

Limita minimă pentru dimensiunea unei stive a unui fir de execuţie este PTHREAD_STACK_MIN.

Pentru a seta/afla dimensiunea stivei unui fir de execuţie se pot utiliza funcţiile :

```
int pthread_attr_setstacksize(pthread_attr_t *tattr, int stacksize);
int pthread_attr_getstacksize(pthread_attr_t *tattr, size_t *size);
```

Pentru a specifica adresa de început a unei stive se poate utiliza funcţia :

```
int pthread_attr_setstackaddr(pthread_attr_t *tattr, void *stackaddr);
```

Windows

Fibre de execuţie

Introducere

Comutarea între firele de execuţie în Windows este costisitoare pentru că presupune trecerea prin kernel-mode. De aceea au fost introduse *fibrele* (fibers), care sunt planificate în user space de către programele care le-au creat.

Fiecare fir de execuţie poate avea mai multe fibre, aşa cum un proces poate avea mai multe fire de execuţie. O fibră se execută în contextul firului de execuţie care o planifică.

Sistemul de operare nu este conştient de existenţa fibrelor, el vede doar firul de execuţie în cadrul căruia fibrele există (o fibră împrumută identitatea firului de care aparţine). De exemplu dacă o fibră execută `ExitThread`, firul care a lansat fibra respectivă se va termina.

O fibră nu are asociate aceleaşi informaţii de stare ca şi firul de execuţie, ci are asociate doar următoarele informaţii de stare :

- stiva
- un subset al regiştrilor firului de execuţie
- datele fibrei furnizate la crearea fibrei

Cum se face planificarea?

Fibrele nu sunt planificate preemptiv (sunt lăsate să ruleze până cedează de buna voie procesorul), cu observaţia că dacă firul din care fac parte este preemptat, automat fibră în execuţie este preemptată. O fibră este planificată printr-o comutare către ea dintr-o altă fibră. Kernelul planifică fire de execuţie, nu fibre.

ConvertThreadToFiber

Înainte de a planifica prima fibră trebuie apelată funcţia `ConvertThreadToFiber` pentru a se crea o zonă în care să se salveze informaţii despre starea fibrei. După executarea acestei funcţii, firul apelant devine fibra activă (fibra în curs de execuţie). Informaţia de stare pentru această primă fibră include datele pasate ca argument funcţiei `ConvertThreadToFiber`.

```
LPVOID ConvertThreadToFiber (LPVOID lpParameter);
```

- `lpParameter` - [in] Pointer la o variabilă pasată fibrei. Fibra va putea obţine aceste date folosind macroul `GetFiberData`.

În caz de succes este returnată adresa fibrei. Altfel, rezultatul este NULL şi tipul erorii se poate afla folosind `GetLastError`.

Există şi funcţia cu efect contrar, `ConvertFiberToThread` :

```
BOOL ConvertFiberToThread(void);
```

Crearea unei fibre

Funcţia `CreateFiber` este folosită pentru a crea o nouă fibră dintr-una deja existentă (deci DUPĂ apelul `ConvertThreadToFiber`).

Aceasta funcţie creează un obiect de tip fibră, îi alocă o stivă şi setează execuţia fibrei să înceapă la adresa de start specificată (funcţia fibrei). Această funcţie NU planifică fibra.

```
LPVOID CreateFiber (
        SIZE_T dwStackSize,
        LPFIBER_START_ROUTINE lpStartAddress,
        LPVOID lpParameter
);
```

- `dwStackSize` - dimensiunea, în bytes, a stivei.
- `lpStartAddress` - pointer la funcţia ce trebuie executată de către fibră.

Atentie! Aceasta nu va fi executată decât după un apel `SwitchToFiber` către fibră.
Această funcţie arata aşa :

```
VOID CALLBACK FiberProc(PVOID lpParameter);
```

unde `FiberProc` ţine locul numelui funcţiei, iar `lpParameter` este un pointer la datele fibrei, este acelaşi cu `lpParameter` transmis funcţiei `CreateFiber`.

- `lpParameter` - pointer la o variabilă pasată fibrei. Fibra va putea obţine aceste date folosind macroul `GetFiberData`.

Numărul de fibre ce poate fi creat de către un fir de execuţie este limitat de către memoria virtuală disponibilă. Implicit, fiecare fibră are 1M rezervat pentru stivă, deci se pot crea cel mult 2028 fibre.

Planificarea unei fibre

Pentru a executa o fibra creată cu `CreateFiber` se foloseşte funcţia `SwitchToFiber`. Aceasta va salva starea fibrei curente şi va restaura starea fibrei specificate. Se poate apela `SwitchToFiber` cu adresa unei fibre create de către un fir de execuţie diferit, pentru aceasta însă trebuie cunoscută adresa returnată celuilalt fir de execuţie de către funcţia `CreateFiber` şi trebuie folosită o sincronizare adecvată.

```
VOID SwitchToFiber(LPVOID lpFiber);
```

`lpFiber` reprezintă adresa fibrei ce este planificată.

Atentie! Apelul :

```
SwitchToFiber(GetCurrentFiber());
```

poate provoca probleme neprevăzute.

Alte funcţii utile

`GetFiberData` poate fi folosită pentru a obţine datele asociate unei fibre (valoarea parametrului `lpParameter` din apelul

uneia din funcţiile `CreateFiber` sau `ConvertThreadToFiber`). Aceeaşi valoare este primită ca parametru de către funcţia asociată fibrei.

```
PVOID GetFiberData(void);
```

`GetCurrentFiber` poate fi utilizată pentru a obţine adresa fibrei, care este chiar rezultatul întors de `CreateFiber` sau `ConvertThreadToFiber`.

```
PVOID GetCurrentFiber(void);
```

Fiber Local Storage

Se poate folosi `Fiber Local Storage` (FLS) pentru a crea o copie unică a unei variabile pentru fiecare fibră. Daca nu apare nici o comutare între fibre FLS se comportă exact ca şi TLS. Funcţiile FLS (`FlsAlloc`, `FlsFree`, `FlsGetValue` şi `FlsSetValue`) manipulează FLS-ul asociat fibrei curente. Dacă firul execută o fibră, şi fibra se schimbă (prin comutare), atunci şi FLS-ul va fi schimbat.

Funcţiile folosite pentru crearea, accesarea şi distrugerea FLS-ului sunt similare cu cele pentru TLS.

Ştergerea unei fibre

Pentru a şterge datele asociate unei fibre se foloseşte funcţia `DeleteFiber`. Aceste date includ stiva, un subset al regiştrilor şi datele fibrei. Dacă o fibră în execuţie apelează `DeleteFiber`, firul asociat ei va apela `ExitThread` şi se va termina. Totuşi, dacă o fibra activă este ştearsă de către o altă fibră, firul care rulează fibra se va termina anormal, pentru că stiva fibrei (care este şi a firului de execuţie) a fost eliberată.

```
void DeleteFiber(LPVOID lpFiber);
```

`lpFiber` specifică adresa fibrei ce va fi ştearsă.