3. Многозадачность в ОС GNU/Linux. Часть 2

Разделы:

- Потоки
- Создание потоков
- Отмена потоков
- Использование потоковых данных

Потоки

- В Linux реализована библиотека функций работы с потоками, соответствующая стандарту POSIX, которая называется *pthreads*

- Все функции и типы данных библиотеки объявлены в файле *pthread.h*

- Они не входят в стандартную библиотеку языка C, поэтому при сборке программы нужно указывать опцию *-lpthread*

Создание потока

```
#include <pthread.h>
int pthread_create
(
 pthread_t* thread,
 const pthread_attr_t *attr,
 void*(*start_routine) (void *),
 void* arg
);
```

Создание потока

```
#include <pthread.h>
#include <stdio.h>
void* printYs(void* unused)
{
  while (!0)
    fputc('y', stderr);
  return NULL;
}
int main()
{
  pthread_t threadId;
  pthread_create(&threadId, NULL, &printYs, NULL);
  while (!0)
    fputc('e', stderr);
  return 0;
}
```

Создание потока

```
#include <pthread.h>
#include <stdio.h>
struct TCharPrintParams
{
  char character;
  int count;
};
void* PrintChars(void* parameters)
{

  struct TCharPrintParams* p = (struct
  TCharPrintParams *) parameters;
  int i;
  for (i = 0; i < p->count; ++i)
    fputc(p->character, stderr);
  return NULL;
}
```

Создание потока

```
int main()
{
  pthread_t thread1Id;
  pthread_t thread2Id;
  struct TCharPrintParams thread1Args;
  struct TCharPrintParams thread2Args;
  thread1Args.character = 'y';
  thread1Args.count = 300;
  pthread_create(&thread1Id, NULL, &PrintChars,
   &thread1Args);
  thread2Args.character = 'e';
  thread2Args.count = 200;
  pthread_create(&thread2Id, NULL, &PrintChars,
   &thread2Args);
  return 0;
}
```

Ожидание завершения потоков

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

struct TCharPrintParams
{
  char character;
  int count;
};
void* PrintChars(void* parameters)
{
  struct TCharPrintParams* p = (struct TCharPrintParams *)
   parameters;
  int i;

  for (i = 0; i < p->count; ++i)
    fputc(p->character, stderr);
  return NULL;
}
```

Ожидание завершения потоков

```
int main()
{
  pthread_t thread1Id;
  pthread_t thread2Id;
  struct TCharPrintParams thread1Args;
  struct TCharPrintParams thread2Args;
  thread1Args.character = 'y';
  thread1Args.count = 300;
  pthread_create(&thread1Id, NULL, &PrintChars,
   &thread1Args);

  thread2Args.character = 'e';
  thread2Args.count = 200;
  pthread_create(&thread2Id, NULL, &PrintChars,
   &thread2Args);
  pthread_join(thread1Id, NULL);
  pthread_join(thread2Id, NULL);
  return 0;
}
```

Значения, возвращаемые потоками

```
#include <pthread.h>
#include <stdio.h>
void* ComputePrime (void* arg)
{
  int candidate = 2;
  int n = *((int*) arg);
  while (!0) {
    int factor;
    int isPrime = 1;
    for (factor = 2; factor < candidate; ++factor)
      if (candidate % factor == 0) {
        isPrime = 0;
        break;
      }
    if (isPrime) {
      if (--n == 0)
        return (void*) candidate;
    }
    ++candidate;
  }
  return NULL;
}
```

Значения, возвращаемые потоками

```
int main()
{
  pthread_t thread;
  int whichPrime = 500;
  int prime;
  pthread_create(&thread, NULL, &ComputePrime, &whichPrime);

  /* Do some other work here...  */
  /* Wait for the prime number thread to complete, and get the result.  */
  pthread_join(thread, (void*) &prime);

  /* Print the largest prime it computed.  */
  printf("The %dth prime number is %d.\n", whichPrime, prime);
  return 0;
}
```

Идентификаторы потоков

- Функция *pthread_self()* возвращает идентификатор потока, в котором она вызвана
- Для сравнения двух разных идентификаторов предназначена функция *pthread_equal()*
- Эти функции удобны для проверки соответствия заданного идентификатора текущему потоку

```
if (0 == pthread_equal(pthread_self(),
    otherThread))
    pthread_join(otherThread, NULL);
```

Атрибуты потоков

- Для задания собственных атрибутов потока выполняются следующие действия:
  1. Создается объект типа *pthread_attr_t*
  2. Вызывается функция *pthread_attr_init()*, которой передается указатель на объект
  3. В объект записываются требуемые значения атрибутов
  4. Указатель на этот объект передается функции *pthread_create()*
  5. Чтобы удалить объект из памяти, вызывается функция *pthread_attr_destroy()*

Атрибуты потоков

```
/*
 * Code snippet for detached thread
 */

#include <pthread.h>

void* threadFunction(void* threadArg)
{
  /* Do work here...  */
  return NULL;
}

int main()
{
  pthread_attr_t attr;
  pthread_t thread;

  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
  pthread_create(&thread, &attr, &threadFunction, NULL);
  pthread_attr_destroy(&attr);

  /* Do work here...  */

  /* No need to join the second thread.  */
  return 0;
}
```

Отмена потока

- Обычно поток завершается при выходе из потоковой функции или в результате вызова функции *pthread_exit()*
- можно запросить *принудительное завершение* или *отмену*
- Чтобы отменить поток, вызывается функция *pthread_cancel()*
- С точки зрения возможности отмены потоки могут быть в одном из трех состояний
  - *Асинхронно отменяемый* в любой точке его выполнения.
  - *Синхронно отменяемый* только в определенных точках выполнения по запросу.
  - *Неотменяемый* (все попытки сделать это игнорируются)

Синхронные и асинхронные потоки

- Чтобы сделать поток асинхронно отменяемым, пользуются функцией *pthread_setcanceltype()*
- Первый аргумент – *PTHREAD_CANCEL_ASYNCHRONOUS* либо *PTHREAD_CANCEL_DEFERRED*
- Второй аргумент – указатель на переменную, куда записывается предыдущее состояние потока
- Делаем поток асинхронным:

```
pthread_setcanceltype
  (PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
```

Неотменяемые потоки

- Поток может отказаться удаляться, вызвав функцию *pthread_setcancelstate()*
- Оказывает влияние только на вызывающий поток
- Первый аргумент *PTHREAD_CANCEL_ENABLE*, если нужно запретить отмену потока, и *PTHREAD_CANCEL_DISABLE* – в противном случае
- Второй аргумент – указатель на переменную, в которой хранится предыдущее состояние потока
- Запрещаем отмену потока:

```
pthread_setcancelstate
   (PTHREAD_CANCEL_DISABLE, NULL);
```

Неотменяемые потоки

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

float* g_AccountBalances;

int ProcessTransaction(int fromAccount, int toAccount, float dollars)
{
  int oldCancelState;

  /* Check the balance in FROM_ACCT.  */
  if (g_AccountBalances[fromAccount] < dollars)
    return 1;

  /* Begin critical section.  */
  pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldCancelState);

  /* Move the money.  */
  g_AccountBalances[toAccount] += dollars;
  g_AccountBalances[fromAccount] -= dollars;

  /* End critical section.  */
  pthread_setcancelstate(oldCancelState, NULL);

  return 0;
}
```

Область потоковых данных

- Все потоковые данные объявляются как переменные типа *void **
- Для создания ключей, т.е. новых переменных, предназначена функция *pthread_key_create()*
- Первым аргументом этой функции является указатель на переменную типа *pthread_key_t*
- Туда записывается значения ключа, с помощью которого любой поток сможет обращаться к своей копии данных
- Второй аргумент – указатель на **функцию очистки ключа**

Область потоковых данных

- После создания ключа любой поток может назначать ему свое значение посредством вызова функции *pthread_setspecific()*
- Первый аргумент – ключ, второй – требуемое значение типа *void* *
- Для чтения данных используется *pthread_getspecific()* с одним аргументом – ключом.

Область потоковых данных

```
#include <malloc.h>
#include <pthread.h>
#include <stdio.h>
static pthread_key_t g_ThreadLogKey;
void WriteToThreadLog(const char* message)
{
  FILE* threadLog = (FILE*) pthread_getspecific(g_ThreadLogKey);
   fprintf(threadLog, "%s\n", message);
}
void CloseThreadLog(void* threadLog)
{  fclose((FILE*) threadLog); }

void* ThreadFunction(void* threadArg)
{
  char threadLogFilename[20];
  FILE* threadLog;
  sprintf(threadLogFilename, "thread%d.log", (int) pthread_self());
  threadLog = fopen(threadLogFilename, "w");
  pthread_setspecific(g_ThreadLogKey, threadLog);
  WriteToThreadLog("Thread starting.");
  /* Do work here... */
  return NULL;
}
```

Область потоковых данных

```
int main()
{
  int i;  pthread_t threads[5];
  pthread_key_create(&g_ThreadLogKey, CloseThreadLog);
  for (i = 0; i < 5; ++i)
    pthread_create(&(threads[i]), NULL, ThreadFunction, NULL);
  for (i = 0; i < 5; ++i)
    pthread_join(threads[i], NULL);
  return 0;
}
```

Область потоковых данных

- Для регистрации обработчика нужно вызвать функцию *pthread_cleanup_push()*
- Ей передается указатель на обработчик и значение аргумента
- Каждому ее вызову должен соответствовать вызов функции *pthread_cleanup_pop()*
- Он отменяет регистрацию обработчика
  - эта функция принимает целочисленный флаг
  - когда он не равен нулю, при отмене регистрации выполняется очистка

Область потоковых данных

```
#include <malloc.h>
#include <pthread.h>
void* AllocateBuffer(size_t size)
{  return malloc(size); }
void DeallocateBuffer(void* buffer)
{  free (buffer); }
void DoSomeWork()
{
  void* tempBuffer = AllocateBuffer(1024);
  pthread_cleanup_push(DeallocateBuffer,
  tempBuffer);
  /* Do some work here that might call
  pthread_exit or might be     cancelled...
  */
pthread_cleanup_pop(1);
}
```

See also

- Робачевский, А. Операционная система Unix, 2 изд./ А.Робачевский, С.Немнюгин, О.Стесик. – СПб.: БХВ-Петербург, 2010. – 656 с.
- POSIX thread (pthread) libraries - http://www.yolinux.com/TUTORIALS/LinuxTutorialP
- Инструменты Linux для Windows-программистов - http://rus-linux.net/nlib.php?name=/MyLDP/BOOKS/Linux-tools/index.html
- Лав, Р. Linux. Системное программирование/ Р.Лав. – СПб.: Питер, 2008. – 416 с.