Потоки управления

- Назначение и свойства потоков управления
- Основные операции над потоками
- Способы реализации потоков
- Работа с потоками управления в ОС Windows
- Работа с потоками управления в ОС Unix
- Потоковая безопасность

Назначение потоков управления

- Мультипроцессные системы с разделением времени позволили одновременное (параллельное) выполнение нескольких программ.
- Каждый процесс имеет собственное, защищенное от вмешательства других адресное пространство — собственные данные.
- Нередко требуется параллельная работа нескольких участков кода с общими данными

Поток управления

- Поток управления (нить, thread) — это элемент процесса, который может выполняться на центральном процессоре.
- Процесс начинает свою работу с одного (главного) потока, но может создавать дополнительные.
- Процесс завершает свое выполнение после завершения работы главного потока (независимо от работы дополнительных потоков)

Поток управления

- Каждый из дополнительных потоков управления описывается функцией, которую называют потоковой функцией.
- При создании потока эта функция начинает выполняться (псевдо)параллельно с основной функцией программы.
- С завершением потоковой функции поток прекращает свою работу

Процесс и потоки

- Все потоки процесса разделяют общие
  - адресное пространство;
  - таблицу открытых файлов;
  - глобальные и статические переменные;
  - другие ресурсы процесса.
- Каждый поток имеет отдельные
  - стек;
  - точку выполнения;
  - значения локальных переменных.

Основные операции над потоками

- Создание потока
  - запуск потоковой функции;
  - передача аргументов в потоковую функцию.
- Завершение работы потока
  - при завершении потоковой функции;
  - по инициативе завершаемого потока;
  - по инициативе другого потока.
- Получение возвращенного потоком значения

Способы реализации потоков

- В ядре системы (ОС Windows)
  - преимущества:
    - планировщик процессора знает о наличии потоков и может учитывать их при планировании процессорного времени;
    - блокировка отдельного потока не оказывает влияния на другие потоки процесса.
  - недостатки:
    - на переход в режим ядра и обратно при выполнении операций над потоками требуется время;
    - введение потоков нередко значительно снижает стабильность ядра ОС.

Способы реализации потоков

- На уровне пользовательских функций (большинство версий Unix)
  - преимущества:
    - операции над потоками выполняются быстрее;
    - не требуют модификации ядра ОС, оно остается простым и, как следствие, стабильным.
  - недостатки:
    - блокировка одного потока может заблокировать весь процесс, т.к. планировщик процессора не знает о наличии в нем других потоков;
    - время процессора делится поровну между процессами без учета количества потоков в них.

Работа с потоками управления ОС Windows

- Основные операции над потоками:
  - создание — `CreateThread`
  - завершение работы — `ExitThread`, `TerminateThread`
  - получения кода завершения потока - `GetExitCodeThread`

Создание потока

  - HANDLE — дескриптор созданного потока
- CreateThread(
  - LPSECURITY_ATTRIBUTES — атрибуты безопасности, обычно NULL,
  - SIZE_T - начальный размер стека, 0 — размер по умолчанию,
  - LPTHREAD_START_ROUTINE — потоковая функция,
  - LPVOID — параметр, передаваемый в потоковую функцию,
  - DWORD — флаги, CREATE_SUSPENDED чтобы поток не начинал выполнение сразу (впоследствии он запускается через ResumeThread),
  - LPDWORD — адрес для получения идентификатора потока, NULL если не нужно

)

Задание

Создайте поток с размером стека по умолчанию, потоковой функцией `StartRoutine`, передавая ей в качестве параметра адрес переменной `Abc`. Поток создается в неактивном состоянии. Дескриптор потока сохраните в переменной `hThread`.

Активируйте созданный поток.

Создание потока

- `HANDLE hThread=CreateThread(NULL, 0,StartRoutine,(LPVOID)&Abc, CREATE_SUSPENDED,0);`
- `...`
- `ResumeThread(hTread);`

Потоковая функция

  - `DWORD` — код завершения потока
  - `WINAPI` — определяет способ вызова функции (calling convention), должен быть указан у всех callback-функций
- `ThreadProc(` - название выбирается пользователем
  - `LPVOID` — параметр потокой функции в виде нетипизированного указателя

  `)`

Завершение работы потока

- Поток завершает свою работу в 4-х случаях:
  - потоковая функция завершила свою работу (код завершения указывается в операторе `return`);
  - в потоке вызвана функция `ExitThread` (с указанием кода завершения);
  - в другом потоке вызвана функция `TerminateThread` (с указанием кода завершения);
  - завершил свою работу процесс, которому принадлежит поток (код завершения потока считается равным коду завершения процесса).

Завершение работы потока

- void - функция не возвращается

- `ExitThread(`
  - DWORD — код завершения потока

  `)`

  - BOOL — 0 если неудачно, иначе не 0
- `TerminateThread(`
  - HANDLE — дескриптор завершаемого потока,
  - DWORD — код завершения потока

  `)`

Задание

Напишите потоковую функцию `FactThread` потока, получающего на вход целое число и возвращающего его факториал в качестве кода завершения.

Потоковая функция

```
DWORD WINAPI FactThread(LPVOID pN)
{
    int N=(int)pN;
    int i,f;
    f=1;
    for(i=2;i<=N;i++)
        f*=i;
    return f;
}
```

Получение кода завершения потока

  - `BOOL` — 0 при неуспешной работе
- `GetExitCodeThread(`
  - `HANDLE` — дескриптор потока, для которого запрашивается код завершения,
  - `LPDWORD` — адрес, по которому будет записан код завершения (или `STILL_ACTIVE`, если поток еще не завершился)

  `)`

Задание

Запустите поток для вычисления факториала числа 10 и получите результат в переменную `fact`.

Получение кода завершения потока

```
int fact;
HANDLE hThread=CreateThread(0,0,
  FactThread,(LPVOID)10,0,0);
/*Ожидаем завершения потока,
  возможно выполняя какие-то
  действия*/
GetExitCodeThread(hThread, &fact);
```

Потоки в ОС Unix

- Концепция потоков появилась позднее создания базового ядра Unix, поэтому реализация потоков в различных версиях ОС различалась.
- При разработке стандарта POSIX были разработаны POSIX-потоки (pthread), поддерживаемые большинством современных unix-систем.
- При компиляции программы с использованием POSIX-потоков необходимо указать ключ `-lpthread`

Операции над потоками в ОС Unix

- Создание (`pthread_create`)
- Завершение (`pthread_exit`)
- Ожидание завершения (`pthread_join`)

Создание потока

  - `int` — положительное значение при ошибке, иначе 0

- `pthread_create(`
  - `pthread_t *` - адрес для записи идентификатора созданного потока,
  - `pthread_attr_r *`— атрибуты, обычно 0,
  - `void* (*start_routine)(void *)` - потоковая функция,
  - `void *` - аргумент потоковой функции

  `)`

Потоковая функция

  - `void *` - возвращаемое значение
- `start_routine(` - имя выбирается пользователем
  - `void *` - параметр потока

  `)`
- Поскольку возвращаемое значение является указателем, то возникает опасность **ошибки**: вернуть адрес *локальной* переменной, которая будет уничтожена при завершении потоковой функции; в результате чего значение по этому адресу потеряется.

Завершение работы потока

- При завершении работы потоковой функции
- При вызове в потоке функции
  - `void pthread_exit(`
    - `void *` - возвращаемое из потока значение

    `)`
- При завершении работы процесса, в котором находится поток.

Ожидание завершения работы потока и получение возвращенного значения

  - `int` — 0 если успешно, иначе положительное значение

- `pthread_join(`
  - `pthread_t` — идентификатор потока, завершения которого ожидаем,
  - `void **` - адрес для записи возвращаемого потоком значения, `NULL` если оно не требуется

)

Задание

- Напишите потоковую функцию `fact_routine`, получающую целую число и возвращающую его факториал.
- Создайте поток для вычисления факториала числа 10 и дождитесь его завершения, запомнив результат в целую переменную `fact`.

Потоковая функция

```
void *fact_routine(void *f)
{
    int x=(int)f;
    int f=1;
    int i;
    for(i=2;i<=x;i++)
        f*=i;
    int *res=(int *)
 malloc(sizeof(int));
    *res=f;
    return (void *)res;
}
```

Работа с потоками

```
int fact;
int *f1;
pthread_t pthr;
pthread_create(&pthr,0,
  fact_routine,(void*)10);
pthread_join(pthr,(void **)&f1);
fact=*f1;
free(f1);
```

Потоковая безопасность

- Функция называется потоково-безопасносной (thread safe), если (псевдо)параллельное выполнение этой функции в нескольких потоках не может нарушить работу программы.
- Функция является потоково-безопасной, если
  - она использует только локальные переменные, и указатели только на выделенную ею память;
  - если работа с глобальнами и статическими переменными, а также получаемыми указателями, синхронизирована;
  - функция может не допускать одновременных запусков, блокируя свою работу с помощью критической секции или мьютекса.

Части CRT, не являющиеся потоково-безопасными

- Библиотека CRT создавалась задолго до появления концепции потоков, поэтому не является потоково-безопасной.
- Небезопасной при многопоточной работе является обработка ошибок, поскольку она использует глобальную переменную `errno`
- Небезопасной при многопоточной работе является, например, функция `strtok`, т.к. для хранения состояния между запусками она использует статические переменные.

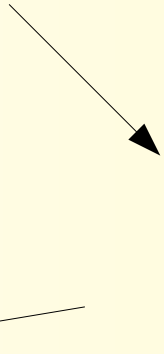
Небезопасность обработки ошибок

Поток 1

- `ff=fopen("file.txt","r");`
-
-
- perror("fopen:");

Поток 2

- `ptr=(int*)malloc(4);`
- `perror("malloc:");`



В этом потоке будет выведен неверный код ошибки