Операционные системы

Лекция № 4

Потоки

(3 часа)

Понятие потока

В традиционных ОС - каждый процесс имеет адресное пространство и один поток управления.

Такая модель процесса базируется на двух независимых концепциях:

- группирование ресурсов
- выполнение программы.

Понятие потока

- С точки зрения выполнения программы процесс – поток исполняемых команд или просто **программный поток (thread**).

- Термин **поток исполнения** (thread) следует отличать от **потока данных** (stream).

- Потоки **разделяют ресурсы процесса** и выполняются **только в рамках** какого-либо процесса. При этом различные потоки могут выполнять **один и тот же** код.

Понятие потока

При запуске процесса создается один единственный поток – **первичный**. В дальнейшем процессы могут создавать дополнительные потоки.

Как только завершается **последний** поток – процесс считается завершенным.

Элементы процесса (общие для всех потоков)	**Элементы потока** (индивидуальные)
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и их обработчики	
Информация об использовании ресурсов	

Многопоточность

Многопоточностью называется способность ОС поддерживать в рамках одного процесса выполнение нескольких потоков.

Противоположный подход - **однопоточный**.

Примеры:

- Один однопоточный процесс: MS-DOS
- Множество однопоточных процессов: разновидности классической UNIX
- Множество многопоточных процессов: OS/2, Linux, Solaris, Mach, Windows NT.

Причины использования потоков

1) Сохранение отзывчивости к командам пользователя, вводу данных и управлению окнами в процессах с GUI;

2) Масштабирование производительности приложения в многопроцессорных и многоядерных ЭВМ;

3) Обеспечение работы приложения, во время приостановки при выполнении I/O;

4) Для реализации фоновой работы в приложениях.

Порождение и завершение потоков

Выполняемая в потоке функция (точка входа в поток) в Linux:

```
void* ThreadFunc(void* arg)
{
 // код потока
};
```

Порождение и завершение потоков

Современная реализация потоков в Linux базируется на стандарте POSIX. Этот интерфейс известен как «pthreads» (POSIX threads).

Основные вызовы относящиеся к этой библиотеке описаны в заголовочном файле `<pthread.h>`

Основной тип, идентифицирующий поток – `pthread_t` (имеет локальный смысл в контексте процесса).

Порождение и завершение потоков

Создание потока:

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

Порождение и завершение потоков

Здесь:

`thread_id` – указатель на идентификатор создаваемого потока (если равен NULL, то не возвращается);

`attr` – аттрибуты потока или NULL (атрибуты по умолчанию);

`start_fn` – функция, содержащая код потока;

`arg` – аргумент, передаваемый функции запуска потока.

Возвращает 0 в случае успеха, иначе – код ошибки.

Порождение и завершение потоков

Пример:

```
// функция, содержащая код потока
void* my_thread(void* arg)
{
    printf("поток № %d работает\n",
 *((int*)arg) );
}
```

Порождение и завершение потоков

Пример:

```
…
pthread_t tid;
int thread_num = 1;
int err = pthread_create(&tid, NULL,
 my_thread, (void*)&thread_num);
if( err != 0) { err_exit(err,"не
 удалось создать поток"); … } else {
… }
```

Порождение и завершение потоков

При сборке многопоточного приложения, использующего библиотеку POSIX Threads (pthread) ее следует явно подключить к программе при сборке, указав компоновщику опцию `-l`. Например:

```
g++   myProgram.cpp  -lpthread  -o
 Run.Me
g++ myProgram.cpp -pthread
```

Порождение и завершение потоков

Завершение потока:

1. Возвратом управления из функции потока (`return`);
2. Вызовом `pthread_exit`;
3. Другим потоком: `pthread_cancel`;
4. Завершение работы процесса

Порождение и завершение потоков

Завершение текущего потока:

```
void pthread_exit(void* status);
```

где `status` – указывает на код возврата из потока;

Отмена указанного потока:

```
int pthread_cancel(pthread_t tid);
```

При удачном завершении возвращают 0, иначе – не нулевое значение, при этом код возврата завершенного потока = `PTHREAD_CANCELED`

```
#include <signal.h>
int pthread_kill(pthread_t tid, int signal);
```

Ожидание потока

Ожидание завершения потока и получение кода возврата:

```
int pthread_join(pthread_t tid, void** status);
```

Действие: блокирует вызывающий поток до тех пор, пока не завершится поток с идентификатором `tid` и помещает его код возврата в переменную `status`.

При удачном завершении возвращает `0`.

Идентификация потоков

Получение идентификатора текущего потока:

```
pthread_t pthread_self (void);
```

Сравнение идентификаторов двух потоков:

```
int pthread_equal (pthread_t TID1,
    pthread_t TID2);
```

Если TID1 и TID2 относятся к одному и тому же потоку, то функция возвращает ненулевое значение, если к разным потокам – 0.

Порождение и завершение потоков. Windows

Определение точки входа в поток:

```
DWORD WINAPI ThreadFunc(LPVOID
  lpThreadParam)
{
… // код потока
return 0;
}
```

`lpThreadParam` – указатель на передаваемые в поток в качестве аргумента данные.

Порождение и завершение потоков

Создание потока и получение его дескриптора:

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpsa,
  DWORD dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpThreadParam,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

Порождение и завершение потоков

Здесь:

`lpsa` – указатель на структуру атрибутов защиты;

`dwStackSize` – размер стека нового потока в байтах (обычно 1 Мб). Значению `0` соответствует размер стека основного потока;

`lpStartAddress` – указатель на функцию – точку входа в поток;

`lpThreadParam` – указатель на аргумент функции потока;

Порождение и завершение потоков

`dwCreationFlags` – задает состояние потока после запуска. Может принимать значения:

- `0` – поток запускается сразу же после создания;
- `CREATE_SUSPENDED` – создание потока в приостановленном состоянии;

`lpThreadId` – указатель на переменную, получающую идентификатор потока.

Порождение и завершение потоков

В случае успеха `CreateThread` возвращает дескриптор созданного потока, иначе – `NULL`.

Порождение и завершение потоков

Пример:

```
// функция – точка входа в поток
DWORD WINAPI MyThread(LPVOID param)
{
 cout << "Hello, my number - "
<< *((int *) param);
    return 0;
}
```

Порождение и завершение потоков

```
…
DWORD ThreadID;
// передаваемый в поток параметр
int ThreadNumber = 1;
HANDLE hThread = CreateThread(NULL, 0,
 MyThread,  (LPVOID)&ThreadNumber, 0,
 &ThreadID);
if (hThread != NULL) { …   }
```

Порождение и завершение потоков

Исполнение потока завершается:

1. Функция потока возвращает управление `return` (лучший вариант);
2. Поток завершает свое выполнение с помощью `ExitThread` (не рекомендуется);
3. Сторонний поток завершает текущий с помощью `TerminateThread` (не рекомендуется);
4. Завершается процесс, запустивший данный поток (не рекомендуется).

Порождение и завершение потоков

Завершение текущего потока:

```
void ExitThread( DWORD dwExitCode );
```

Принудительное завершение указанного потока:

```
BOOL TerminateThread( HANDLE hThread,
 DWORD dwExitCode );
```

Получение кода возврата указанного потока:

```
BOOL GetExitCodeThread( HANDLE
 hThread, LPDWORD lpExitCode );
```

Идентификация потоков

- `GetCurrentThread` – возвращает псевдодескриптор текущего потока;
- `GetCurrentThreadId` – возвращает идентификатор текущего потока;
- `GetThreadId` – возвращает идентификатор потока по известному дескриптору;
- `OpenThread` – создает дескриптор потока по известному идентификатору.

Порождение и завершение потоков

Некоторые функции CRT могут приводить к небольшой утечке памяти при работе с потоками.

Кроме WinAPI вызовов для работы с потоками в C/C++ существуют функции:

```
_beginthreadex();
_endthreadex();
```

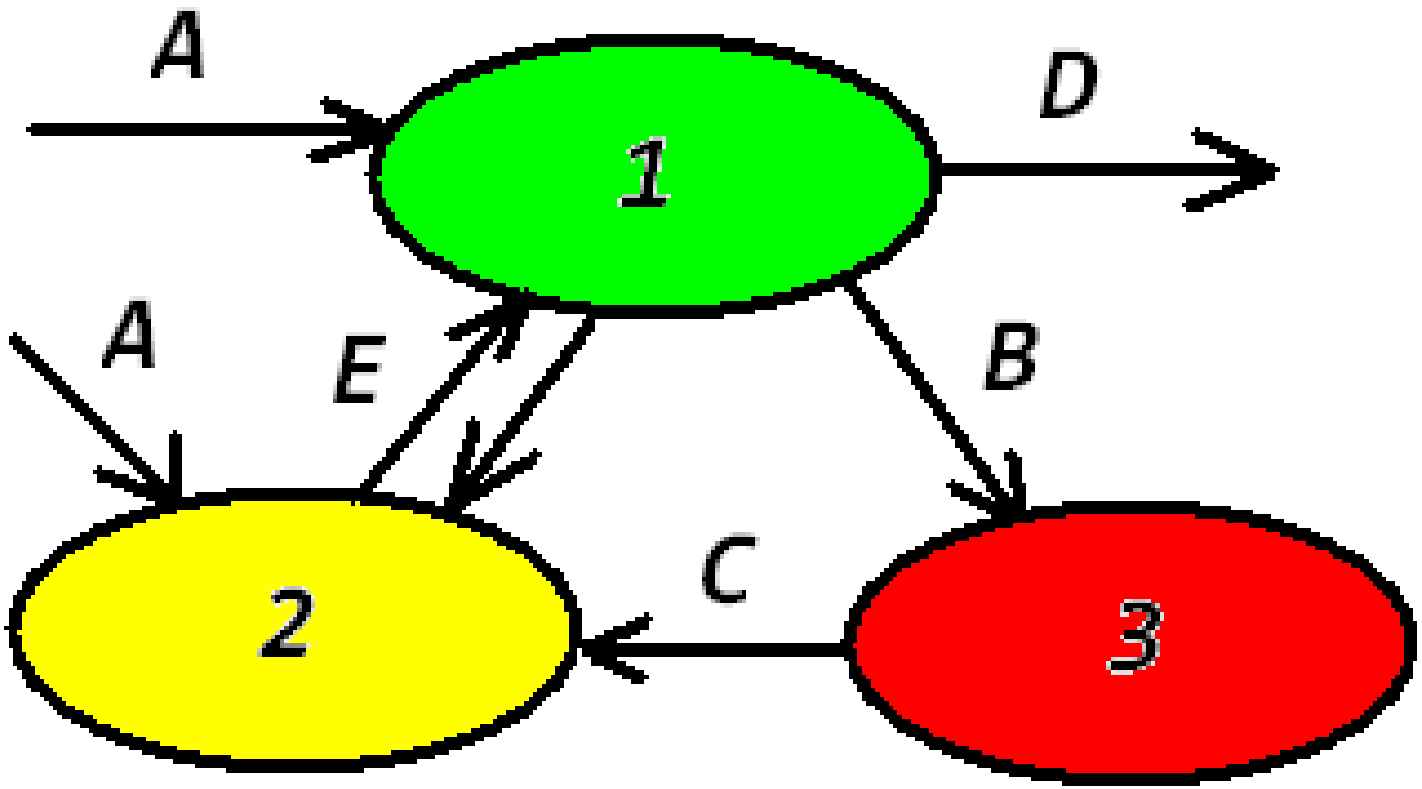
позволяющие вызывать такие функции корректно.

Состояния потоков

Основные состояния потоков:

- Состояние выполнения потока (running state) – поток фактически выполняется процессором;
- Состояние готовности (ready state) – поток может выполняться;
- Состояние блокировки(wait state) (так же говорят о блокированных (blocked) или спящих (sleeping) потоках) – поток выполняет функцию ожидания несигнализирующих объектов или объектов синхронизации, завершения операции I/O и т.д.

Состояния потоков



Состояния потоков

Основные действия с потоками:

A. Порождение;
B. Блокирование;
C. Разблокирование – в таком случае говорят, что поток «пробуждается» (wakes);
D. Завершение.
E. Выбор планировщиком следующего потока – по истечению кванта времени или при вызове `Sleep(0)`.

Не существует способа, позволяющего программе определить состояние другого потока.

Состояния потоков

Для перевода потока в приостановленное состояние используется:

```
DWORD SuspendThread (HANDLE hThread);
```

Для возобновления выполнения используется:

```
DWORD ResumeThread (HANDLE hThread);
```

Оба вызова принимают дескриптор управляемого потока. Возвращают предыдущее значение счетчика приостановок в случае, если завершаются успешно, или иначе - `0xFFFFFFFF`.

Ожидание в течение конечного интервала времени

`VOID Sleep (DWORD dwMilliseconds);`

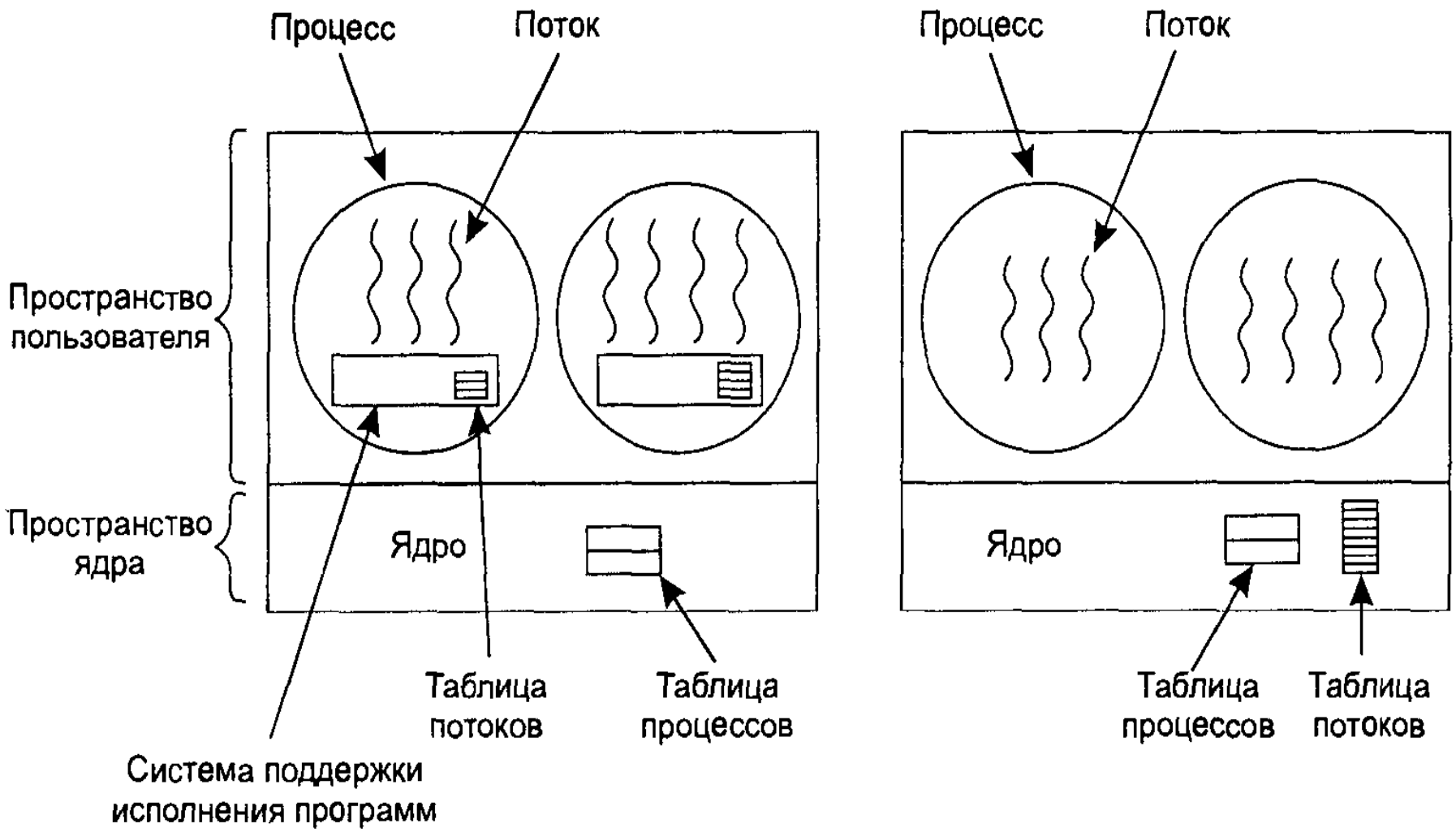
позволяет потоку отказаться от CPU и перейти из состояния выполнения в состояние ожидания, которое будет длится в течение заданного промежутка времени. По истечении периода ожидания планировщик переводит поток в состояние готовности.

Здесь,

`dwMilliseconds` – интервал ожидания в милисекундах, `INFINITE` – бесконечный промежуток ожидания или `0` – отказ потока от оставшейся части отведенного кванта времени.

Реализация потоков

- **В пространстве пользователя** (user-level threads – **ULT**)
  примеры: P-Threads от POSIX, C-Threads от Mach
- **На уровне ядра** (kernel-level threads – **KLT**)
  примеры: OS/2, Linux, Windows NT.
- Смешанные решения (Solaris)


Процесс
Поток
Пространство пользователя
Пространство ядра
Ядро
Таблица потоков
Таблица процессов
Система поддержки исполнения программ
Процесс
Поток
Ядро
Таблица процессов
Таблица потоков

ULT. плюсы:

1. Более высокая производительность. Переключение потоков не включает в себя переход в режим ядра.
2. Планирование производится в зависимости от специфики приложений.
3. Использование потоков на пользовательском уровне применимо для любой ОС.

минусы:

1. В типичной ОС многие системные вызовы являются блокирующими. При этом ОС блокирует все потоки процесса.
2. Приложение только с ULT не может воспользоваться преимуществами многопроцессорной системы.

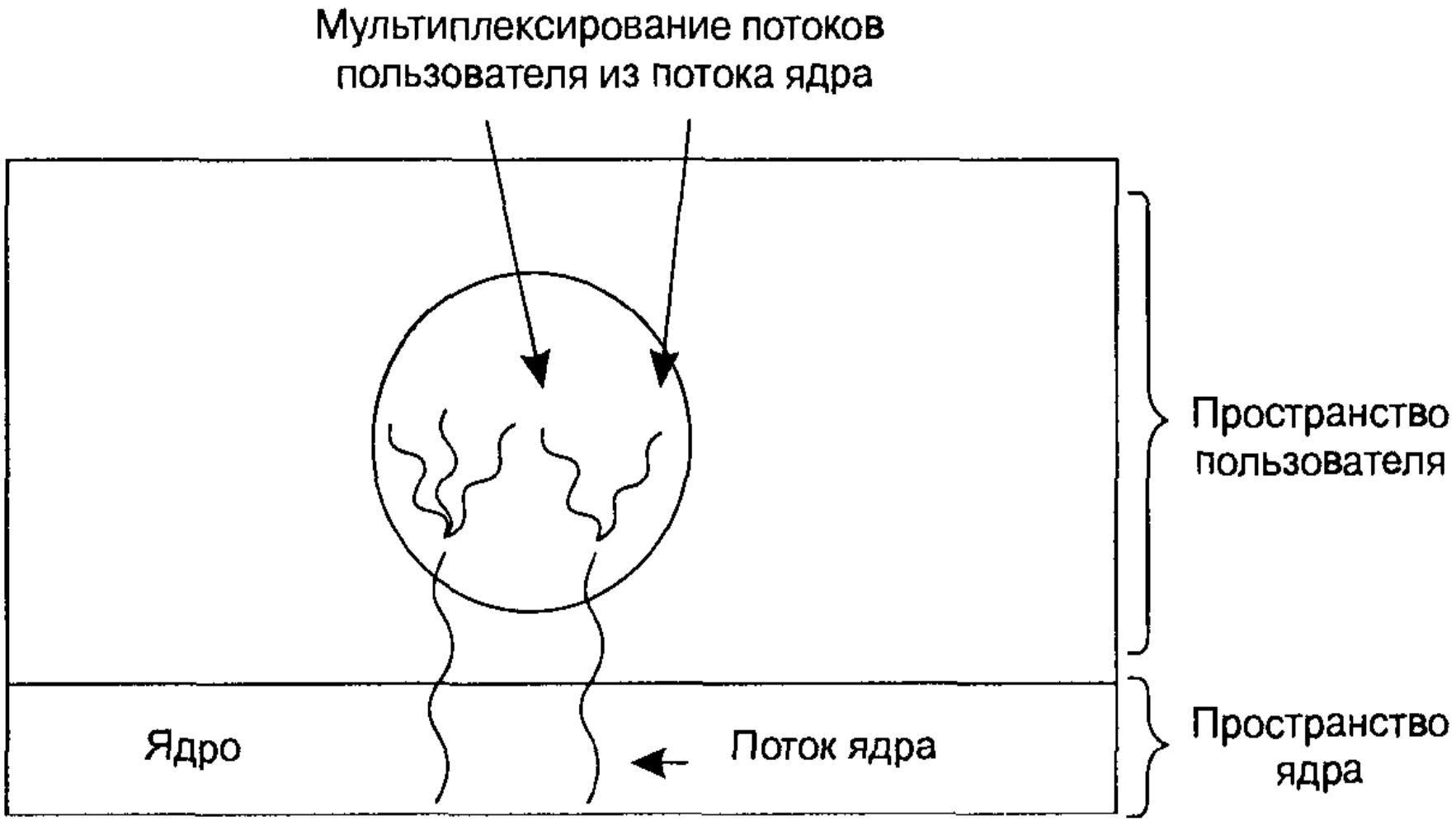
KLT. плюсы:

1. Простота реализации.
2. Блокировка только вызвавшего блокирующий запрос потока. Отсутствие необходимости проверки произойдет ли блокировка при выполнении системного вызова.
3. Возможность использовать преимущества многопроцессорной системы.

минусы:

1. Существенно большие затраты на системные вызовы создания, планирования и завершения потоков.

Смешанные решения



Плюсы использования потоков

1. Создание нового потока в уже существующем процессе занимает намного меньше времени, чем создание совершенно нового процесса.
2. Поток можно завершить намного быстрее, чем процесс.
3. Переключение потоков в рамках одного и того же процесса происходит намного быстрее.
4. При использовании потоков повышается эффективность обмена информацией между двумя выполняющимися программами.

Плюсы использования потоков применительно к программированию

- Одновременная работа в приоритетном и фоновом режимах
- Асинхронная обработка
- Высокая скорость выполнения
- Модульная структура программы

Недостатки потоков

- Потоки усложняют программную модель
- Потоки совместно используют данные – требуется синхронизация
- Сложность отладки и обработки ошибок