Ожидание завершенных дочерних процессов

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);

int WIFEXITED(int status);
int WIFSIGNALED(int status);
int WIFSTOPPED(int status);
int WIFCONTINUED(int status);

int WEXITSTATUS(int status);
int WTERMSIG(int status);
int WSTOPSIG(int status);
int WCOREDUMP(int status);
```

Ожидание завершенных дочерних процессов

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
	int status;
	pid_t pid;

	if(!fork())
		return 1;
	pid = wait(&status);
	if(pid == -1)
		perror("wait");
	printf("pid=%d\n", pid);
```

Ожидание завершенных дочерних процессов

```
    if(WIFEXITED(status))
        printf("Нормальное завершение, статус=%d\n",
            WEXITSTATUS(status));
    if(WIFSIGNALED(status))
        printf("Прерывание по сигналу=%d%s\n",
            WTERMSIG(status),
            WCOREDUMP(status)?"(dumped core)":"");
    if(WIFSTOPPED(status))
        printf("Остановлен сигналом=%d\n",
            WSTOPSIG(status));
    if(WIFCONTINUED(status))
        printf("Продолжение процесса\n");
    return 0;
}
```

Ожидание завершенных дочерних процессов

```
[user@user-pc OS]$ ./wait_test
pid=2512
Нормальное завершение, статус=1
```

Ожидание завершенных дочерних процессов

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
```

pid	
< -1	ожидание любого дочернего процесса, чей ID группы процессов равен абсолютному значению этой величины
-1	ожидание любого дочернего процесса
0	ожидание любого дочернего процесса, принадлежащего той же группе, что и вызывающий
> 0	ожидание любого дочернего процесса, чей pid в точности равен указанной величине

Ожидание завершенных дочерних процессов

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
```

options	
WNOHANG	Не блокировать вызов, немедленно вернуть результат, если ни один процесс не завершился
WUNTRACED	Устанавливает параметр WIFSTOPPED, даже если вызывающий процесс не отслеживает свой дочерний
WCONTINUED	Устанавливает WIFCONTINUED, даже если вызывающий процесс не отслеживает свой дочерний

Потоки (Threads)

- Процесс содержит один или несколько потоков выполнения
- Процесс с одним потоком представляется одной структурой **`task_struct`**
- Многопоточный процесс имеет по одной структуре на каждый поток
- С точки зрения ядра Linux не существует отдельной концепции потоков
- Потоки – просто способ совместного использования ресурсов несколькими процессами

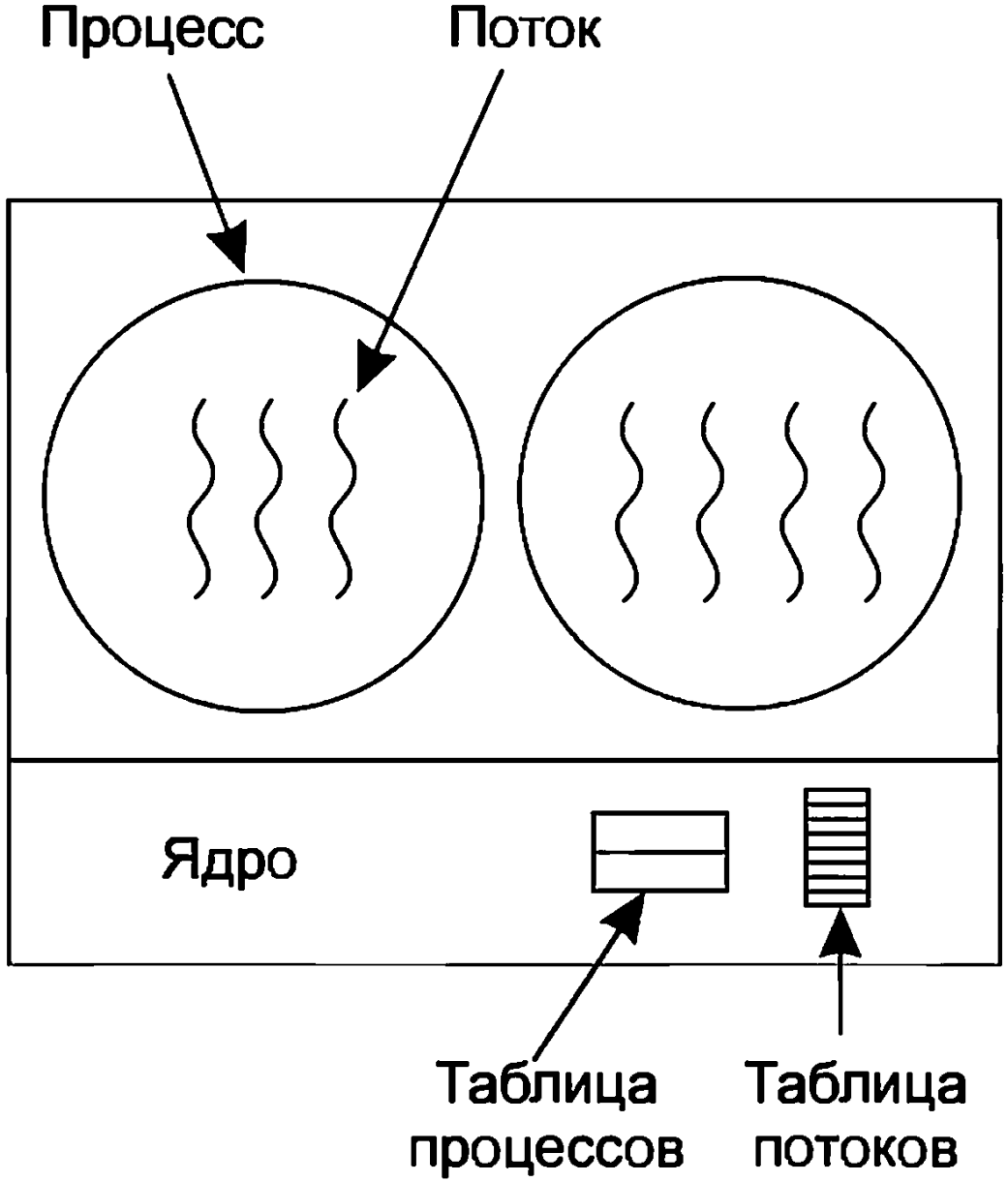
Преимущества многопоточности

- Ускорение реакции процесса на действия пользователя
- Блокировка при операциях ввода-вывода
- Время, затрачиваемое на переключение контекста
- Экономия памяти
- Параллелизм

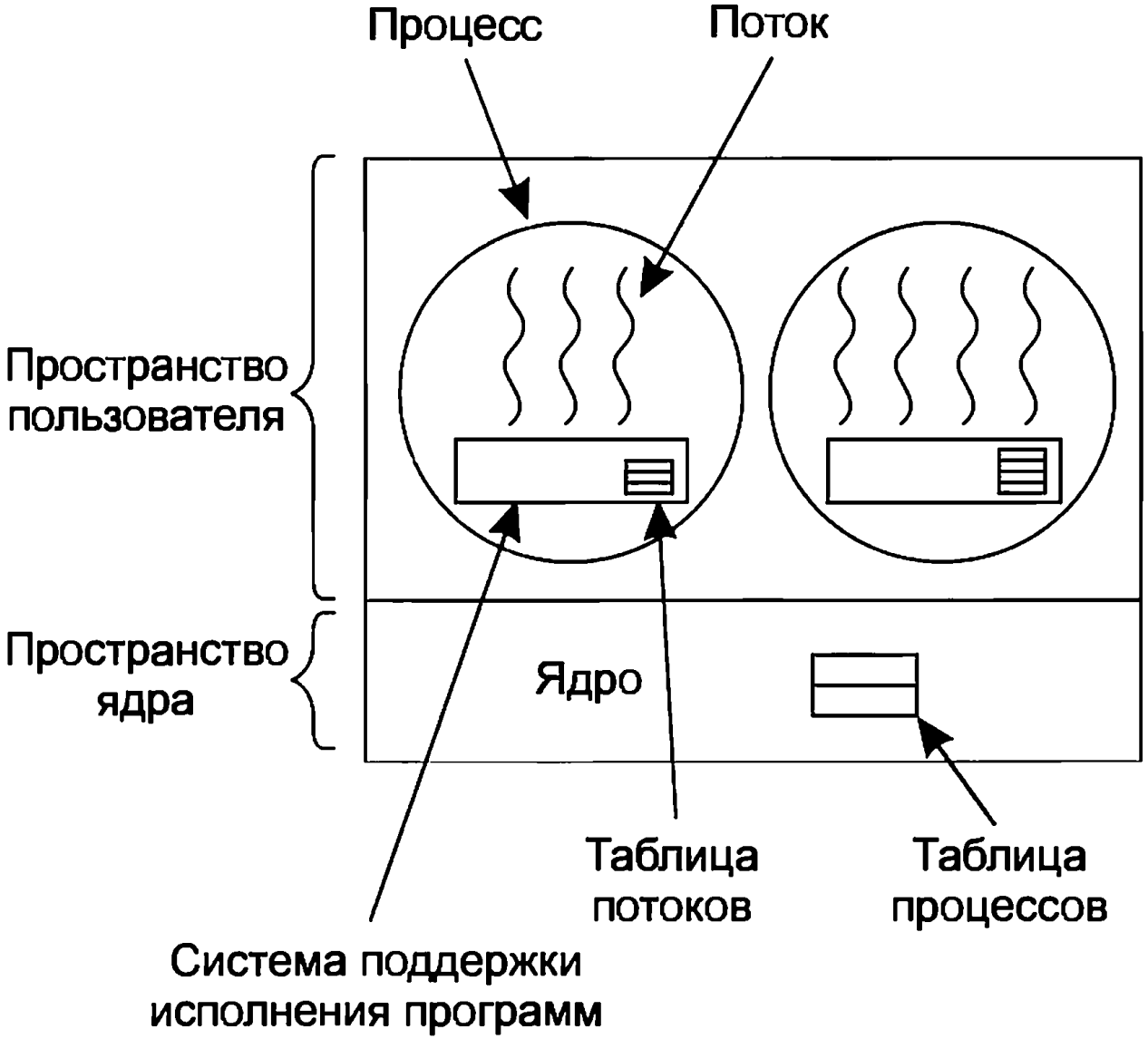
Использование объектов потоками

Свойства, присущие каждому процессу	Свойства, присущие каждому потоку
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и обработчики сигналов	

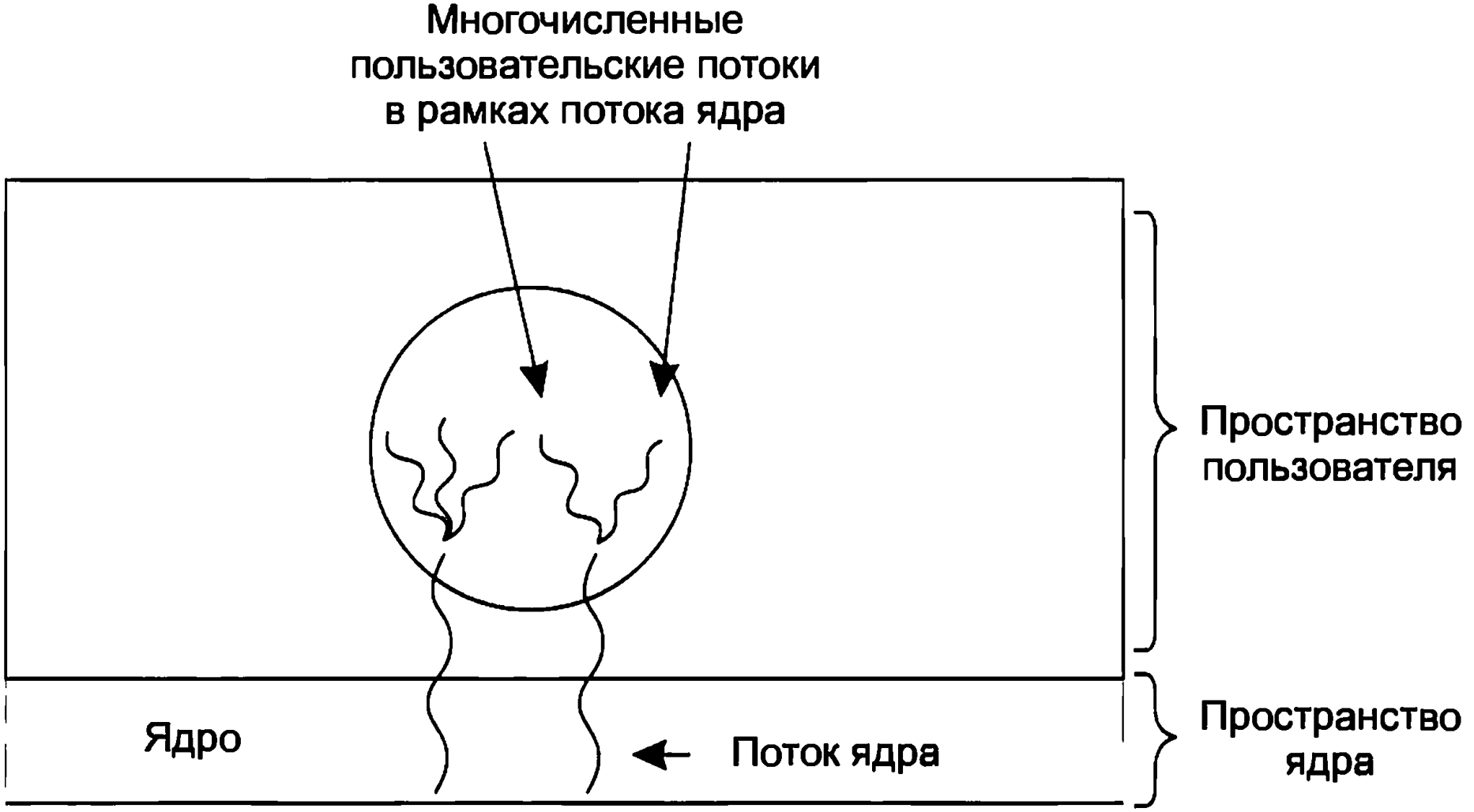
Реализация потоков на уровне ядра



Реализация потоков на уровне пользователя



Гибридная реализация поточности



POSIX-потоки (P-потоки)

- *LinuxThreads* – оригинальная реализация P-потоков, обеспечивающая поточность на уровне ядра (glibc 2.0)

- *Native POSIX Thread Library (NPTL)* заменила *LinuxThreads* и остается стандартной реализацией P-потоков в Linux (ядро 2.6, glibc 2.3)

API для работы с P-потоками

```
#include <pthread.h>

int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);

void *start_thread (void *arg);
```

Пример

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

void * thread_func(void *arg)
{
      int i;
      int loc_id = * (int *) arg;
      for (i = 0; i < 4; i++){
            printf("Thread %i is running\n", loc_id);
            sleep(1);
      }
}
```

Пример (продолжение)

```
int main(int argc, char * argv[])
{
      int id1, id2, result;
      pthread_t thread1, thread2;
      id1 = 1;
      result = pthread_create(&thread1, NULL, thread_func,
                                                &id1);
      if (result != 0) { return EXIT_FAILURE; }

      id2 = 2;
      result = pthread_create(&thread2, NULL, thread_func,
                                                &id2);
      if (result != 0) { return EXIT_FAILURE; }
```

Пример (продолжение)

```
    result = pthread_join(thread1, NULL);
    if (result != 0) { return EXIT_FAILURE; }

    result = pthread_join(thread2, NULL);
    if (result != 0) { return EXIT_FAILURE; }

    printf("Done\n");
    return EXIT_SUCCESS;
}
```

Пример (окончание)

При наличии функции pthread_join	Без функции pthread_join

```
[user@user-pc pth]$ gcc pth.c -lpthread
[user@user-pc pth]$ ./a.out
```

```
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Done
```

```
Done
```

API для работы с P-потоками

```
#include <pthread.h>


int pthread_join(pthread_t thread,
                 void **retval);
```

API для работы с P-потоками

```
#include <pthread.h>


int pthread_exit(void *retval);
```