Управление процессами, часть 6

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение, 2019

Содержание лекции

Потоки

Потоки (threads)

позволяют одновременно выполнять разные действия в контексте одной программы.

Механизмы работы с потоками реализованы в *Linux* в отдельной библиотеке *Pthread*.

В рамках лекции будут рассмотрены следующие темы:

- Понятие потоков и их особенности.
- Создание и завершение потока.
- Синхронизация потоков.
- Получение информации о потоках.
- Обмен данными между потоками.

Когда один процесс вызывает fork(), в системе появляется другой процесс, который выполняет тот же код. Но данные, которыми манипулирует этот код, являются независимой копией данных процесса-родителя.

```
#include <stdio.h>
#include <unistd.h>
int main (void){
  int a = 10;
  pid_t status = fork ();
  if (!status) {
    a++;
    printf ("Child's a=%d", a);
    return 0;
  }
  wait ();
  printf ("Parent's a=%d", a);
  return 0;
}
```

Процессы...

- ... могут иметь общие данные, но для этого программисты обращаются к методам межпроцессного взаимодействия (interprocess communication).
- Межпроцессное взаимодействие подчас требует сложных манипуляций со стороны программиста.
- В некоторых случаях плохо реализованное межпроцессное взаимодействие может стать мишенью для злоумышленников.

Потоки

- ...позволяют в рамках одной программы выполнять одновременно несколько действий, используя при этом общие данные.
- ...выполняются так же, как и процессы, т. е. независимо

Создание потока в Linux

1. Создается функция, которая называется потоковой функцией.
2. При помощи функции pthread_create() создается поток, в котором начинает параллельно остальной программе выполняться потоковая функция.
3. Вызывающая сторона продолжает выполнять какие-то действия, не дожидаясь завершения потоковой функции.

Чтобы подключить библиотеку Pthread к программе, нужно передать компоновщику опцию -lpthread.

Создание потока в Linux

1. Создается функция, которая называется потоковой функцией.
2. При помощи функции pthread_create() создается поток, в котором начинает параллельно остальной программе выполняться потоковая функция.
3. Вызывающая сторона продолжает выполнять какие-то действия, не дожидаясь завершения потоковой функции.

Чтобы подключить библиотеку Pthread к программе, нужно передать компоновщику опцию -lpthread.

- каждый поток имеет идентификатор типа *pthread_t*
- идентификаторы потока существуют локально в рамках текущего процесса.

Создание потока в Linux

```
int pthread_create (pthread_t * THREAD_ID, void * ATTR,
                    void *(*THREAD_FUNC) (void*), void * ARG);
```

Аргументы функции:

- THREAD_ID - идентификатор нового потока (если таковой был создан).
- ATTR - указание атрибутов потока. Если этот аргумент равен NULL, то поток создается с атрибутами по умолчанию.
- PTHREAD_FUNC - указатель на потоковую функцию. Это обычная функция, возвращающая бестиповый указатель (void*) и принимающая бестиповый указатель в качестве единственного аргумента.
- ARG — указатель, содержащий аргументы потока. Если потоковая функция не требует наличия аргументов, то в качестве ARG можно указать NULL.

Пример nothread.c

- программа при запуске создает один поток, а функция pthread_create() позволяет создавать дополнительные потоки;
- основную программу следует трактовать как «родительский поток»;
- если один из потоков завершает программу, то все остальные потоки тут же завершаются.

```
void * any_func (void * args) {
  fprintf (stderr, "Hello World");
  sleep (5);
  return NULL;
}
int main (void){
  any_func (NULL);
  fprintf (stderr, "Goodbye World");
  while (1);
  return 0;
```

Пример thread1.c

- any_func() теперь - потоковая функция;
- функции main() и any_func() работают параллельно, никакой видимой задержки между выводом двух сообщений не происходит.

```
void * any_func (void * args) {...}
int main (void){
  pthread_t thread;
  int result;
  result = pthread_create(&thread, NULL, &any_func, NULL);
  if (result != 0) {
     fprintf (stderr, "Error");
     return 1;
  }
  fprintf (stderr, "Goodbye World");
  while (1);
  return 0;
}
```

Пример makefile для thread1.c

```
thread1: thread1.c
	gcc -o $@ $^ -lpthread
clean:
	rm -f thread1
```

Пример threadargs.c

- Для передачи данных в поток используется четвертый аргумент функции pthread_create();
- Этот указатель автоматически становится аргументом потоковой функции.

```
void * any_func (void * arg)
{
    int a = *(int*) arg;

    fprintf (stderr, "Hello World with argument=%d", a);

    return NULL;
}
```

Пример threargs.c

```
int main (int argc, char ** argv)
{
    pthread_t thread;
    int arg, result;
    if (argc < 2) {
        fprintf (stderr, "Too few arguments");
        return 1;
    }
    arg = atoi (argv[1]);
    result = pthread_create (&thread, NULL, &any_func, &arg);

    fprintf (stderr, "Goodbye World");
    while (1);
    return 0;
}
```

Пример threadstruct.c

- Если требуется передать в поток несколько аргументов, то их можно разместить в структуре, указатель на которую также передается в потоковую функцию.

```
struct thread_arg {
    char * str;
    int num;
};
void * any_func (void * arg)
{
    struct thread_arg targ = (struct thread_arg *)arg;
    fprintf (stderr, "str=%s", targ->str);
    fprintf (stderr, "num=%d", targ->num);

    return NULL;
}
```

Пример threadstruct.c

```
int main (void){
    pthread_t thread;
    int result;

    struct thread_arg targ;
    targ.str = "Hello World";
    targ.num = 2007;

    result = pthread_create (&thread, NULL, &any_func, &targ);

    while (1);
    return 0;
}
```

Пример threadexit.c

Потоки могут завершаться:

- возвратом из потоковой функции;
- вызовом специальной функции void pthread_exit(void * RESULT).

Если потоковая функция вызывает другие функции, то pthread_exit() в таких случаях бывает очень полезной.

```
void print_msg (void) {
    fprintf (stderr, "Hello World");
    pthread_exit (NULL);
}
void * any_func (void * arg)
{
    print_msg ();
    fprintf (stderr, "End of any_func()");
    return NULL;
}
```

Пример threadexit.c

```
int main (void)
{
    pthread_t thread;

    if (pthread_create (&thread, NULL, &any_func, NULL) != 0)
    {
        fprintf (stderr, "Error");
        return 1;
    }

    while (1);
    return 0;
}
```

Ожидание потока: pthread_join()

Функция pthread_join() позволяет синхронизировать потоки:

```
int pthread_join (pthread_t THREAD_ID, void ** DATA);
```

- функция блокирует вызывающий поток до тех пор, пока не завершится поток с идентификатором THREAD_ID.
- по адресу DATA помещаются данные, возвращаемые потоком через функцию pthread_exit() или через инструкцию return потоковой функции.
- при удачном завершении pthread_join() возвращает 0, любое другое значение сигнализирует об ошибке.

Пример join1.c

```
#define A_COUNT 15
#define B_COUNT 10
void * print_b (void * arg)
{
  int i;
  for (i = 0; i < B_COUNT; i++) {
    fprintf (stderr, "B");
    sleep (1);
  }
  fprintf (stderr, "C");
  return NULL;
}
```

Пример join1.c

```
int main (void){
    pthread_t thread; int i;
    if (pthread_create (&thread, NULL, &print_b, NULL) != 0) {
      fprintf (stderr, "Error");
      return 1;
    }
    for (i = 0; i < A_COUNT; i++) {
      fprintf (stderr, "A");
      sleep (1);
    }
    if (pthread_join (thread, NULL) != 0) {
      fprintf (stderr, "Join error");
      return 1;
    }
    fprintf (stderr, "D");
    return 0; }
```

Пример join1.c

Эта программа в течение некоторого времени выводит примерно такой "шифр":

```
$ ./join1
ABABABABABABABABABABACAAAAD
```

- Символы A печатает родительский поток.
- Символы B выводятся порожденным потоком. Когда этот поток завершается, выводится символ C.
- Когда завершается программа, печатается символ D.
- Кроме того, применение функции pthread_join() наконец-то избавило нас от необходимости каждый раз запускать бесконечный цикл в конце программы.

Пример join2.c

Пример получения данных, возвращаемых из потока.

```
void * any_func (void * arg)
{
  int a = *(int *) arg;
  a++;
  return (void *) a;
}
```

Пример join2.c

Пример получения данных, возвращаемых из потока.

```
int main (void)
{
  pthread_t thread;
  int parg = 2007, pdata;
  if (pthread_create (&thread, NULL,
          &any_func, &parg) != 0) {
    fprintf (stderr, "Error");
    return 1;
  }
  pthread_join (thread, (void *) &pdata);
  printf ("%d", pdata);
  return 0;
}
```

Пример join3.c

Функцию pthread_join() может вызывать любой поток, работающий внутри текущего процесса.

```
#define A_COUNT 10
#define B_COUNT 25
#define C_COUNT 10
void * print_b (void * arg)
{
  int i;
  for (i = 0; i < B_COUNT; i++) {
    fprintf (stderr, "B");
    sleep (1);
  }
  fprintf (stderr, "(end-of-B)");
  return NULL;
}
```

Пример join3.c

```
void * print_c (void * arg)
{
  pthread_t thread = * (pthread_t *) arg;
  int i;

  for (i = 0; i < C_COUNT; i++) {
    fprintf (stderr, "C");
    sleep (1);
  }

  fprintf (stderr, "(end-of-C)");
  pthread_join (thread, NULL);

  return NULL;
}
```

Пример join3.c

```
int main (void){
  pthread_t thread1, thread2; int i;
  if (pthread_create (&thread1, NULL, &print_b, NULL) != 0)
    return 1;
  if (pthread_create (&thread2, NULL, &print_c, &thread1) != 0)
    return 1;
  for (i = 0; i < A_COUNT; i++) {
    fprintf (stderr, "A");
    sleep (1);
  }
  fprintf (stderr, "(end-of-A)");
  pthread_join (thread2, NULL);
  fprintf (stderr, "(end-of-all)");
  return 0;
}
```

Получение информации о потоке: pthread_self(), pthread_equal()

Это функция pthread_self() возвращает идентификатор текущего потока.

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

Тип pthread_t является целым числом, но к идентификаторам потоков не рекомендуется применять математические операции. Для сравнения идентификаторов двух потоков служит функция pthread_equal():

```
int pthread_equal (pthread_t THREAD1, pthread_t THREAD2);
```

- Если идентификаторы относятся к одному потоку, то эта функция возвращает ненулевое значение.
- Если THREAD1 и THREAD2 являются идентификаторами разных потоков, то pthread_equal() возвращает 0.

Пример join4.c

Чаще всего значение идентификатора потока необходимо для того, чтобы уберечь поток от вызова phtread_join() для самого себя.

```
void * any_func (void * arg)
{
  pthread_t thread = * (pthread_t *) arg;

  if (pthread_equal(pthread_self(), thread) != 0)
    fprintf (stderr, "1");

  return NULL;
}
```

Пример join4.c

```
int main (void)
{
  pthread_t thread;
  if (pthread_create (&thread, NULL,
      &any_func, &thread) != 0) {
    fprintf (stderr, "Error");
    return 1;
  }
  if (pthread_equal (pthread_self(), thread) != 0)
    fprintf (stderr, "2");
  pthread_join (thread, NULL);
  return 0;
}
```

Отмена потока: pthread_cancel()

Любой поток может послать другому потоку запрос на завершение. Такой запрос называют отменой потока:

```
int pthread_cancel (pthread_t THREAD_ID);
```

- Функция pthread_cancel() возвращает 0 при удачном завершении, ненулевое значение сигнализирует об ошибке.
- Несмотря на то, что pthread_cancel() возвращается сразу, это вовсе не означает немедленного завершения потока.
- В связи с этим, если для вас важно, чтобы поток был удален, нужно дождаться его завершения функцией pthread_join().

Пример pcancel1.c

```
void * any_func (void * arg)
{
  while (1) {
    fprintf (stderr, ".");
    sleep (1);
  }

  return NULL;
}
```

Пример pcancel1.c

```
int main (void) {
  pthread_t thread;
  void * result;

  if (pthread_create (&thread, NULL, &any_func, NULL) != 0)
    return 1;

  sleep (5);
  pthread_cancel (thread);

  if (!pthread_equal (pthread_self (), thread))
    pthread_join (thread, &result);
  if (result == PTHREAD_CANCELED)
    fprintf (stderr, "Canceled");
  return 0;
}
```

Дополнительная информация

Не рассмотренные темы можно изучить на следующих страницах справочного руководства man:

- man pthread_atfork;
- man pthread_kill;
- man pthread_once;
- man pthread_mutex_lock; man pthread_mutex_unlock;
- man pthread_detach;
- man pthread_setcancelstate; man pthread_setcanceltype; man pthread_testcancel;
- man pthread_setspecific.

По приведенным далее ссылкам размещены наиболее полные руководства и источники дополнительной информации о потоках:

- http://yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html;
- http://www.llnl.gov/computing/tutorials/pthreads/;
- http://www.ibiblio.org/pub/Linux/docs/faqs/Threads-FAQ/html/;
- http://www.humanfactor.com/pthreads/pthread-tutorials.htm