Лабораторная работа по курсу "Операционные системы"

Потоки в ОС Linux

Цель работы: знакомство с системными вызовами для управления потоками в ОС Linux.

1. Функции для работы с потоками

Прототипы функций работы с потоками и необходимые типы данных содержатся в заголовочном файле **<pthread.h>**. Эти функции не включены в стандартную библиотеку языка С, они находятся в библиотеке **libpthread**. Поэтому при компоновке в командную строку добавьте опцию **−lpthread** (см. лаб. работу 2), например: **$gcc -o pr1 pr1.c -lpthread**

Создание потока. Поток создается функцией **pthread_create (&tid, attr, func, arg)**, где

&tid – указатель на переменную типа **pthread_t**, в нее будет записан ID нового потока.

attr – указатель на атрибуты потока для управления деталями функционирования потока. Если параметр равен NULL, то поток будет создан с атрибутами по умолчанию.

func – указатель на функцию потока, которая принимает один параметр типа указатель на **void** и возвращает значение типа указатель на **void**.

arg – аргумент потока типа **void***. Это значение передается потоку как аргумент в функцию потока. Через него можно передать новому потоку параметры.

Функция **pthread_create** возвращает ноль при успехе и не ноль при ошибке. После создания каждый поток выполняет *функцию потока* **func(arg)** – обычную функцию в программе пользователя. При завершении этой функции поток завершается. Возврат из функции **pthread_create** происходит немедленно, и исходный поток продолжает выполнение команд, следующих за вызовом **pthread_create**. Одновременно новый поток начинает выполнение функции потока.

Завершение потока. При нормальных условиях поток завершается двумя способами:

1. Обычный возврат из функции потока. Величина, возвращаемая **return**, будет являться значением, возвращаемым потоком.
2. Возврат при помощи функции **pthread_exit**. Она может быть вызвана из любой функции данного потока. Аргумент этой функции будет являться значением, возвращаемым потоком.

Передача данных в поток. Т.к. типом аргумента, передаваемого в поток (четвертый параметр функции **pthread_create**), является **void*,** то для передачи одного параметра типа **int** его можно преобразовать к типу **(void*)**. Для передачи большего количества параметров аргумент потока должен быть указателем на область данных или структуру, содержащую передаваемые параметры. Необходимо, чтобы данные, передаваемые новому потоку, были ему доступны. При этом не следует передавать стековые переменные, если поток, вызывающий функцию **pthread_create**, может завершиться ранее, чем созданный поток.

Ожидание завершения потока и возврат значения из потока. В случае потоков аналогом функции **wait** является функция **pthread_join**: поток, вызвавший эту функцию, будет ожидать завершения указанного потока. Функция возвращает ноль в случае нормального выполнения, и не ноль в случае ошибки. Функция имеет два параметра: ID потока, завершения которого следует ожидать, и указатель на **void***, в указатель будет записано значение, возвращаемое потоком. Если это значение не требуется, то второй параметр функции **pthread_join** может быть NULL. В качестве значения, возвращаемого потоком, не следует передавать адреса стековых переменных потока, т.к. после его завершения они будут недоступны.

Пример создания потока, передачи данных в поток и возврата значения из потока:

```
#include <pthread.h>
#include <stdio.h>
double b;
/* Функция потока */
void *thread(void *varp){
  b = *(int*)varp/2.0;      /* Вычисление результата b = 5/2.0 */
```

```
  pthread_exit((void*)&b); /* Значение, возвращаемое потоком, -
                       указатель на переменную b, содержащую результат */
}

void main(){
  int v1=5;
  double *c1;
  pthread_t tid;
  pthread_create(&tid, NULL, thread, (void*)&v1); /* Указатель на
          переменную v1 передан в поток в качестве аргумента потока */
  pthread_join(tid, (void**)&c1); /* В переменную c1 будет записано
          значение, возвращаемое потоком (указатель на результат) */
  printf("%lf\n",*c1);                /* Будет напечатано '2.5' */
}
```

Задание 1. Напишите следующую программу в соответствии со своим вариантом. Распечатайте в каждом потоке ID потока (функции **pthread_self** и **gettid**) и ID процесса (функция **getpid**). Определите ID созданных потоков также при помощи команды **ps**. *Сравните перечисленные выше ID и запишите их в отчете. Выясните и запишите в отчете 1) сколько потоков и процессов создано программой? 2) когда завершается процесс: при завершении главного потока или при завершении всех потоков процесса? 3) остается ли в системе информация о завершенных потоках, если другой поток процесса не вызывает функцию **pthread_join**? 4) функция **sleep**, вызванная в потоке, приостанавливает выполнение процесса или потока?*

Варианты 1, 5, 9, 13, 17, 21, 25. Программа запускается с командной строкой, содержащей два параметра: слово *A* и целое число *k*. Главный поток создает два потока, в первый поток передает *A* и *k*, во второй – *k*. Первый поток печатает слово *A* *k* раз и возвращает число напечатанных символов. Второй поток вычисляет $\sum_{i=1}^{k} i$ и возвращает вычисленное значение. Главный поток печатает значения, возвращаемые потоками.

Варианты 2, 6, 10, 14, 18, 22, 26. Главный поток создает *N* потоков, выполняющих одну и ту же функцию потока, в которую передает число k_i, $i = 1, \ldots, N$. (Число k_i вводится с клавиатуры.) Функция потока вычисляет $k_i!$ и возвращает вычисленное значение в виде вещественного числа. Главный поток печатает сумму полученных значений.

Варианты 3, 7, 11, 15, 19, 23, 27. Введите с клавиатуры числа *m* и *n*. Главный поток создает *m* потоков, в которые передает адрес *i*-ой строки матрицы *m*×*n* и *n* – число столбцов матрицы. Функция потока вычисляет сумму элементов строки и возвращает вычисленное значение. Главный поток вычисляет сумму полученных из потоков значений и печатает ее.

Варианты 4, 8, 12, 16, 20, 24, 28. Главный поток в цикле считывает с клавиатуры строку, создает поток и передает в него строку. Функция потока создает файл с именем *file<ID_потока>.dat*, записывает в него полученную строку и возвращает код возврата: ноль – при успехе, -1 – при неудаче. Главная программа, в зависимости от кода возврата, печатает "*успешно*" или "*неуспешно*".

2. Мьютексы в потоках

Если несколько потоков модифицируют значение общей переменной, то возникает состояние ***гонки***. Решение проблемы заключается в том, чтобы в каждый момент времени доступ к такому участку программы (так называемой ***критической секции***, ***CS***) имел только один поток. Это можно сделать при помощи ***мьютексов*** (*mutex*, сокращение от *MUTual Exclusion*).

Мьютекс – это специальный вид семафора, блокировка, которую в каждый момент времени может устанавливать один поток. Чтобы создать мьютекс, нужно объявить переменную типа **pthread_mutex_t** и присвоить ей начальное значение PTHREAD_MUTEX_INITIALIZER. Перед началом CS следует вызвать функцию **pthread_mutex_lock()**, передав ей параметр – указатель

на мьютекс. В конце CS следует вызвать функцию **pthread_mutex_unlock()**, передав ей параметр – указатель на мьютекс.

Если мьютекс свободен, т.е. ни один из потоков не находится в CS, то при выполнении функции **pthread_mutex_lock()** мьютекс переходит во владение данного потока и из функции **pthread_mutex_lock()** происходит немедленный возврат. Если же мьютекс уже был захвачен другим потоком, то выполнение функции **pthread_mutex_lock()** блокируется и возобновляется только тогда, когда мьютекс вновь становится свободным. Сразу несколько потоков могут ожидать освобождения мьютекса. Когда это событие наступает, только один из потоков (выбираемый произвольно) разблокируется и получает возможность захватить мьютекс. Остальные потоки остаются заблокированными. Функция **pthread_mutex_unlock()** освобождает мьютекс. Она должна вызываться только из того потока, который захватил мьютекс.

Задание 2.

1. Изучите работу программы **mutex.c** (см. Приложение), находящуюся в каталоге данной лабораторной работы. Скопируйте файл **mutex.c** в ваш каталог.
2. Выполните программу **mutex**. Проверьте, что переменная **common** изменяет свою величину от 0 до 100, при этом каждый поток изменяет эту переменную ровно 50 раз. Обратите внимание, что в каждый момент времени переменная **common** читается, увеличивается на 1 и затем записывается без прерывания только одним потоком. Это обеспечивается механизмом мьютексов, который используется обоими потоками (см. функции **pthread_mutex_lock()** и **pthread_mutex_unlock()**).
3. Затем удалите механизм мьютексов из программы **mutex.c**, перекомпилируйте ее и выполните несколько раз. Одинаковы ли значения переменной **common** в разных запусках? Всегда ли переменная **common** имеет конечное значение, равное 100? *Объясните результаты в отчете.*

Порядок выполнения лабораторной работы

1. Выполните задания 1–2.
2. Занесите в отчет описание заданий с требуемыми пояснениями и ответами на вопросы.

Требования

1. При подготовке к лабораторной работе (дома) подготовьте конспект, содержащий а) краткие сведения о функциях работы с потоками; б) текст программы из задания 1. Текст программы должен быть прокомментирован.
2. Студент должен знать ответы на следующие вопросы:

Вопросы

1. Что такое поток?
2. Какова цель использования потоков в программах?
3. Какова разница между потоками и процессами?
4. Когда потоки могут выполняться действительно параллельно?
5. Как можно создать в программе новый поток?
6. Как передать в поток параметры и получить из потока возвращаемое значение?
7. Может ли функция, выполняемая потоком, быть описана вне программы?
8. Могут ли два потока использовать одну и ту же функцию?
9. Что произойдет с потоком, если поток, создавший его, завершится?
10. Возможно ли выполнить многопоточную программу на многопроцессорной системе?
11. Возможно ли выполнить многопоточную программу на нескольких компьютерах, объединенных в сеть?
12. Что такое критическая секция в многопоточной программе?

Источники информации

1. Митчел М., Оулдем Дж., Самьюэл А. Программирование для Linux. Профессиональный подход. – М.: Издательский дом "Вильямс", 2003. (Глава 4) (The original book (2001) is available at http://www.newriders.com or http://www.advancedlinuxprogramming.com)
2. Конспект лекций.

Приложение. Текст программы **mutex.c**

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

void do_one_thing(int *);
void do_another_thing(int *);
void do_wrap_up(int);
int common = 0; /* A shared variable for two threads */
pthread_mutex_t mut=PTHREAD_MUTEX_INITIALIZER;

main()
{
  pthread_t       thread1, thread2;

  if (pthread_create(&thread1, NULL, (void *)do_one_thing, (void *)&common)!=0)
      perror("pthread_create"),exit(1);
  if (pthread_create(&thread2,NULL,(void *)do_another_thing,(void *)&common)!=0)
      perror("pthread_create"),exit(1);
  if (pthread_join(thread1, NULL) != 0)   perror("pthread_join"), exit(1);
  if (pthread_join(thread2, NULL) != 0)   perror("pthread_join"), exit(1);
  do_wrap_up(common);
  return 0;
}

void do_one_thing(int *pnum_times)
{
  int i, j, x;
  unsigned long k;
  int work;
  for (i=0;i<50;i++){
      pthread_mutex_lock(&mut);
      printf("doing one thing\n");
      work = *pnum_times;
      printf("counter = %d\n", work);
      work++; /* increment, but not write */
      for (k=0;k<500000;k++); /* long cycle */
      *pnum_times = work; /* write back */
      pthread_mutex_unlock(&mut);
  }
}
void do_another_thing(int *pnum_times)
{
  int i, j, x;
  unsigned long k;
  int work;
  for (i=0;i<50;i++) {
      pthread_mutex_lock(&mut);
      printf("doing another thing\n");
      work = *pnum_times;
      printf("counter = %d\n", work);
      work++; /* increment, but not write */
      for (k=0;k<500000;k++);  /* long cycle */
      *pnum_times = work; /* write back */
      pthread_mutex_unlock(&mut);
  }
}
void do_wrap_up(int counter)
{
  int total;
  printf("All done, counter = %d\n", counter);
}
```

© Лабораторная работа подготовлена Л.В. Илюшечкиной, А.Е.Костиным (часть 2)