Прикладные физико-технические и компьютерные методы исследований

Семинар 5

https://dl.dropboxusercontent.com/u/96739039/infa_s05.pdf

Средства коммуникации процессов

- Сигнальные
- Канальные (***потоковая модель*** (pipe+работа с файлами), сообщения)
- Разделяемая память

FIFO – именованный pipe

- Pipe позволяет общаться **только родственным** процессам благодаря тому, что таблица файловых дескрипторов (часть системного контекста) наследуется дочерним процессом.
- Аналог pipe для общения неродственных процессов – FIFO
- Данные о расположении и состоянии канала связи FIFO можно узнать через файловую систему (записаны в файле на диске)
- Пример: osstud/05/stud/05-4c.htm

Особенности open для FIFO

- Без дополнительных флагов … при открытии FIFO на запись вызов open блокируется, пока кто-то ещё не откроет её на чтение
- И наоборот, при открытии FIFO на чтение вызов open блокируется, пока кто-то не откроет её на запись

Упражнение 1

- Создать текстовый мессенджер, позволяющий двум неродственным процессам отправлять сообщения друг другу через FIFO
- Для считывания строки из терминала используйте fgets. 3м параметром передавайте stdin

Комментарии к упражнению 1

- **fgets** для считывания текста из терминала блокирующий + **read** для считывания из fifo – тоже блокирующий (как и для pipe’a) … и как с этим работать? …
- не забывайте про особенность open для fifo
- выводите на экран всю отладочную информацию: имена fifo, которые пытаемся создать/открыть, номера соответствующих дескрипторов и т.д.
- код для клиента №0 и клиента №1 очень похож -> можно обойтись одним файлом с кодом … но запуск клиентов производить так:

./a.out 0

и

./a.out 1

- mknod вернёт ошибку, если файлы для работы с fifo уже были созданы ранее. Проверить данный случай можно с помощью кода

```
if (errno != EEXIST)
{
   ….
}
```

Указатели на функции

```
typedef int (*MyFunc)(int, char);
int g(int x, char c) {
        return x + c;
}
void f(MyFunc func) {
        int result;
        result = func(42, ‘z’);
        …
}
int main() {
        f(g);
}
```

Позволяют лишний раз не дублировать код программы.

Упражнение 2

- Реализовать ф-ю сортировки массива, принимающую на вход ф-ю сравнения: с помощью одной ф-и сортировки и двух разных ф-й сравнения отсортировать массив по возрастанию/убыванию.

Прикладные физико-технические и компьютерные методы исследований

Семинар 5 (продолжение)

Пример задачи на многопоточность

Есть N маленьких (~единиц Mpixels) картинок. Нужно применить к ним фильтр, например, blur.

На каждую задачу по отдельному процессу. Процесс = своё адресное пространство (в 1ю очередь).

Есть **одна** очень большая фотография, которую нужно noblur’ить.

Разбиваем её на куски, и каждый будет обрабатывать отдельная нить. Нить = логический процессор.

Нити исполнения (threads)(легковесные процессы)

- Разделяют
  - Программный код (пользовательский)
  - Глобальные переменные (пользовательский)
  - Все нити живут в одном адресном пространстве – «куча» общая (пользовательский)
  - Системные ресурсы (таблица открытых файлов, текущая директория и т.п.)
- Но каждая нить имеет собственный
  - Программный счётчик
  - Содержимое регистров
  - Стек

  т.е. нити могут работать параллельно, и контекст нитей переключается быстрее.

Нити исполнения (threads)

- Общие глобальные переменные + «куча» – можно использовать для коммуникации между нитями
- В Linux posix thread`ы реализованы не эффективно, поэтому выигрыш в производительности можно и не заметить

Нити исполнения (threads)

- ***void* thread(void* arg);***
- **pthread_create** (fork)
- **pthread_exit** (exit)
- **pthread_join** (waitpid)
- В случае ошибки возвращается число > 0 (ранее всегда < 0 обозначало ошибку)

См. пример

https://dl.dropboxusercontent.com/u/96739039/s07e01.c

gcc main.c ***-lpthread***

Завершение работы нити исполнения

- ***pthread_exit*** – завершается вызвавшая ф-ю нить
- возврат из ф-и, связанной с нитью
- возврат из ***main*** или вызов ***exit*** в каком-либо месте программы: при завершении процесса убиваются все его нити -> не забываем про ***pthread_join***

Нити исполнения (threads)

- **int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);**
- **void pthread_exit(void *status);**
- **int pthread_join (pthread_t thread, void **status_addr);**

Упражнение 3

- Модифицируйте предыдущий пример, добавив в него 3-ю нить исполнения

Упражнение 4 (Состояние гонки)

- Создаём массив из 3-х переменных. Создаём две нити.
- 1-я в цикле 10^7 раз увеличивает на единицу a[0] и a[2]
- 2-я в цикле 10^7 раз увеличивает на единицу a[0] и a[2]
- Основная нить дожидается завершения обеих
- Выводим a[0], a[1], a[0] + a[1], a[2]

Упражнение 5

- В мессенджере, который вы реализовали с помощью FIFO замените процессы (для одновременной работы fgets и read) на нити.

Как не надо передавать аргументы в функцию потока...

```
struct Segment {
  int begin;
  int end;
  int index;
};

void* FindAverage(void* arg) {
  struct Segment segment = *((struct Segment*)arg); // (*)
  // ...
}

int main() {
  // ...
  for (int segmentIndex = 0; segmentIndex < segmentsCount; ++segmentIndex)
  {
    struct Segment segment;
    /* Так делать нельзя, т.к. вы передаёте в ф-ю указатель
    // на переменную, созданную на стеке. К моменту копировая в строке (*)
    // в этом месте может быть записано что-то другое. Нужно создать вне
    // цикла массив segments, время жизни которого заведомо больше времени жизни
    // нитей, либо воспользоваться примитивами синхронизации, о которых будем на
    // следующих семинарах говорить.
    */

    // ...
    result = pthread_create(&thread_ids[segmentIndex],
     (pthread_attr_t *)NULL,
     FindAverage,
     &segment);
  }
  // ...
}
```

Упражнение 5 (домашнее: одно из 2х) (необходимо сдать через неделю)

1. Найти произведение двух больших квадратных матриц с помощью N потоков. Размеры матрицы кратны N.
2. Есть большое количество экспериментальных данных (нужно самим сгенерировать)

$\sim 10^8$ вещественных чисел. Необходимо с помощью N потоков найти их среднее и дисперсию.

a)С помощью утилиты time измерить время работы программы в зависимости от N, либо ф-и time

https://rtfm.co.ua/unix-utilita-time-opcii-i-primery-ispolzovaniya/

b) Компилируем с ключом –O2 или -O3: gcc main.c –O2

(получается release версия, которая можно работать в ~10 раз быстрее при том же исходном коде).

c) Ускорение (отношение времени работы в 1 поток и N потоков) напишите в комментариях в самом начале кода. Если на виртуалке ускорения не наблюдается, то запустите там, где Linux – основная ОС + возьмите матрицу, либо кол-во экспериментальных данных побольше. Во второй задаче время генерации данных может быть сравнимо со скоростью счёта. Поэтому замеряйте только время вычисления среднего и дисперсии (без времени генерации данных) с помощью

http://stackoverflow.com/questions/5248915/execution-time-of-c-program