Московский Физико-Технический Институт
Физтех-школа Аэрокосмических технологий
Институт Аэромеханики и Летательной Техники

МФТИ

ФАКТ

# Архитектура Компьютера и Операционные Системы. Часть 2. Основы операционных систем

# Лекция 6: Нити исполнения. Средства IPC: Семафоры

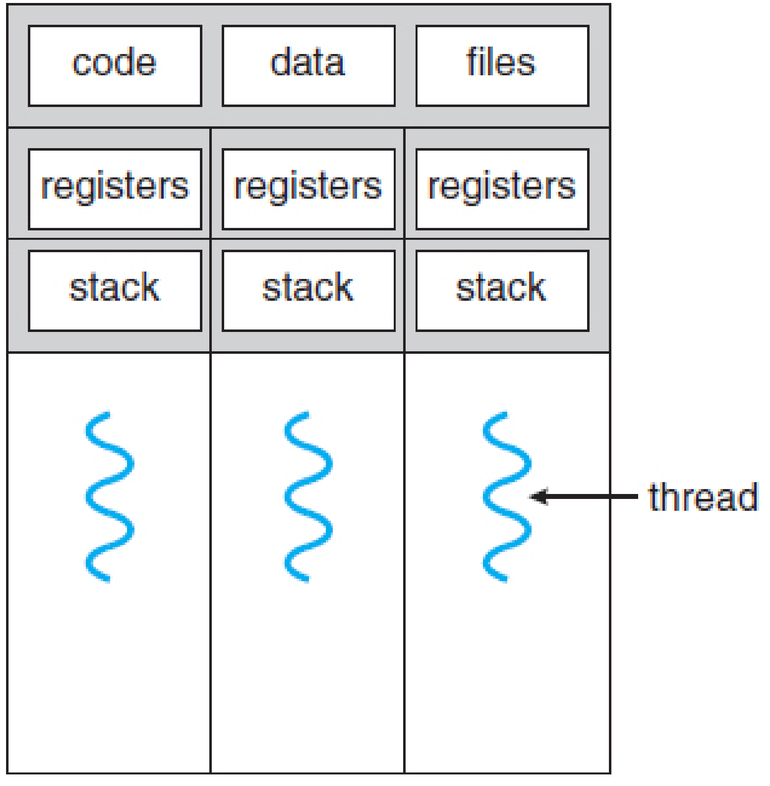
Новиков Андрей Валерьевич
д.ф.-м.н.

Жуковский

# Нити исполнения / потоки выполнения / threads

- Процесс = совокупность выделенных ресурсов + набор нитей исполнения
- Нить – «подпроцесс», «лёгкий процесс»

- Нити процесса **совместно используют**
  - программный код;
  - глобальные переменные;
  - системные ресурсы
- Каждая нить имеет **отдельные**:
  - программный счётчик;
  - содержимое регистров;
  - стек




multithreaded process

# Реализации потоков выполнения

- 1. «Лёгкие» единицы планирования ядра – **потоки выполнения ядра**
  - BSD: Light  Weight Kernel Threads (LWKT);
  - Linux: Native POSIX Thread Library (NPTL);
  - MacOS: Apple Multiprocessing Services
  - Windows: WinAPI

- 2. Потоки в режиме пользователя на уровне библиотек - **пользовательские потоки выполнения**
  - GNU Portable Threads;
  - Thread Manager компании Apple;
  - Windows 7 user-mode scheduling;
  - …

# Стандарт POSIX для потоков выполнения — pthreads

- POSIX threads -> PThreads. (POSIX = **P**ortable **O**perating **S**ystem **I**nterface)
- Стандарт описывает API, реализация не ограничивается. Реализован во всех Unix-подобных ОС и Windows

- Функции POSIX API:
  - pthread_create(), pthread_exit(), pthread_cancel(), pthread_self() – создание, удаление
  - pthread_join(), pthread_detach() – ожидание
  - pthread_attr_init(), pthread_attr_setdetachstate(), pthread_attr_destroy() – аттрибуты;
  - pthread_mutex_init(), pthread_mutex_destroy(), pthread_mutex_lock(), pthread_mutex_trylock(), pthread_mutex_unlock() – синхронизация.
  - pthread_cond_init(), pthread_cond_signal(), pthread_cond_wait() – условные перменные

# Создание потока выполнения

```
#include <pthread.h>

int pthread_create(
   pthread_t* thrid,
   const pthread_attr_t* attr,
   void* (*start)(void*),
   void* arg
);
```

- Возвращаемое значение
  =0 – успех
  >0 – (!) код ошибки, как в errno, но которая не изменяется
- thrid – идентификатор созданной нити.
- attr – атрибуты нити
  NULL – по-умолчанию
- void* start(void*) – функция, которая будет выполняться внутри нити
- arg – аргумент для функции

# Завершение потока выполнения

Варианты завершения потока:

- ❑ Возврат из функции нити
- ❑ Вызов функции pthread_exit()
- ❑ Завершение всего процесса по exit()

```
#include <pthread.h>


void pthread_exit(void* status);
```

# Ожидание потока выполнения

```
#include <pthread.h>

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

Блокирует вызывающий поток до завершения указанной нити

- Возвращаемое значение
  =0 – успех
  >0 – (!) код ошибки, как в errno, но которая не изменяется
- status – указатель, возвращаемый нитью, NULL, если результат не интересует

# Пример с потоком выполнения

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

void pause() {  time_t start_time = time(NULL);  while(time(NULL) == start_time) {} }

void* thread_func(void*) {
     for (int i = 0; i < 20; i++) {
          fputs("b\n",stderr);    pause();
     }
}

int main (int argc, char *argv[], char *envp[]) {
     pthread_t thread;
     if( pthread_create(&thread, NULL, thread_func, NULL) ) return EXIT_FAILURE;
     for(int i = 0; i < 20; i++) {
          puts("a");    pause();
     }

     if ( pthread_join(thread, NULL)) return EXIT_FAILURE;

     return EXIT_SUCCESS;
}
```

# Особенности доступа к совместно используемой памяти

- Совместно используемая память:
  - для потоков исполнения – все глобальные переменные программы
  - для процессов – сегменты SysV IPC shared memory

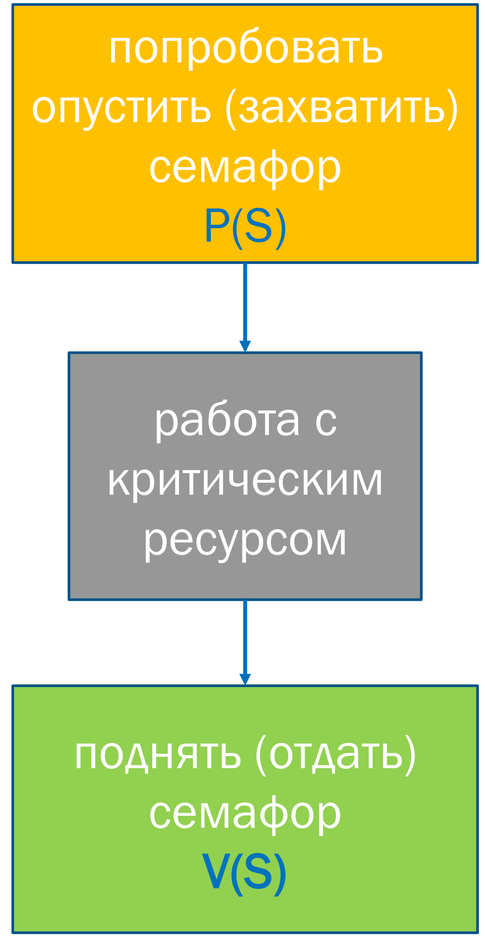
- Одновременное чтение из общей памяти:
  - ОК

- Одновременная запись в общую память:
  - вероятность повреждения
  - необходимость синхронизации

# Синхронизация с помощью семафоров (semaphores)

- Semaphore – примитив синхронизации на основе счётчика
- Предложен Edsger Wybe Dijkstra в 1965.

- Действия над семафором S, атомарные:
  - **P**(S) – ждать, если S==0, затем S = S – 1 (proberen – датск. проверить)
  - **V**(S) – увеличить S = S + 1 (verhogen – датск. увеличивать).



# Семафоры System V IPC

- Расширенный набор операций относительно Dijkstra.

- Действия над семафором SysV IPC, атомарные:
  - A(S, n) – увеличить S = S + n
  - D(S, n) – попытаться уменьшить: ждать пока S < n, затем S = S – n
  - Z(S) – ждать, если S != 0

# Получение доступа / создание семафоров SysV IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(
    key_t key,
    int nsems,
    int semflg
);
```

Получает доступ или создаёт массив семафоров

- Возвращаемое значение
  >0 – дескриптор SysV IPC
  -1 – ошибка, код ошибки в **errno**
- **key** – уникальное имя (ключ), созданное **ftok()** или *IPC_PRIVATE*
- **nsems** – количество семафоров в массиве
- **semflg** – флаги, комбинируются с помощью **|** «битовое или»

# Получение доступа / создание семафоров SysV IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(
   key_t key,
   int nsems,
   int semflg
);
```

- **semflg** – флаги, комбинируются с помощью **|** «битовое или»
  - IPC_CREAT - создать если не существует
  - IPC_EXCL - вместе с предыдущим, создавать эксклюзивно, ошибка если сущуствует
  - Права доступа
  - 0400 – только чтение для владельца
  - 0200 – только запись для владельца
  - 0100 – только исполнение для владельца
  - 0040 – только чтение для группы
  - …

# Операции над семафором SysV IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(
   int semid,
   struct sembuf* sops,
   size_t nsops
);
```

Выполняет операцию над одним или несколькими семафорами

- Возвращаемое значение
  0 – успех
  -1 – ошибка, код ошибки в **errno**
- **semid** – идентификатор массива семафоров
- **sops** – массив операций над несколькими семафорами, не обязательно всеми(!)
- **nsops** – количество операций в массиве sops

# Операции над семафором SysV IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

struct sembuf {
    unsigned short sem_num;
    short          sem_op;
    short          sem_flg;
};
```

Кодирование операции

- sem_num – номер (индекс) семафора в массиве семафоров
- sem_op – код операции над семафором:
  - +n – операция A(S,n)
  - –n – операция D(S, n)
  - 0 – операция Z(S)
- sem_flg – флаги операции, 0 – по-умолчанию

# Порядок действий для работы с семафорами

- 1. Определить с какими семафорами (их индексы) из массива будем работать. Пусть массив из 10 семафоров, а операции будет делать над 2мя
- 2. Подготовить массив `sops` из 2 элементов `struct sembuf`.
- 3. Заполнить элементы массива `sops` кодами нужной операции
- 4. Выполнить системный вызов semop

- semop возвращается, только после выполнения всех операций
  - порядок не определён
  - если одни операции блокирующие, то неблокирующие могут ожидать, а могут выполнится до

# Пример работы с семафорами

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main(int argc, char* argv[], char* envp[])
{
    key_t k = ftok(“mylabel”, 0);  if( k < 0 ){ … }
    int semid = semget(k, 1, IPC_CREAT|0664);  if( semid < 0 ){ … }

    struct sembuf sops[1];
    sops[0].sem_num = 0;  sops[0].sem_op = +1;  sops[0].sem_flg = 0;
    if( semop(semid, sops, 1) < 0 ){ … }
    // …
    return 0;
}
```

# Пример работы с семафорами

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main(int argc, char* argv[], char* envp[])
{
    key_t k = ftok(“mylabel”, 0);  if( k < 0 ){ … }
    int semid = semget(k, 1, IPC_CREAT|0664);  if( semid < 0 ){ … }

    struct sembuf opAs0 = { .sem_num = 0, .sem_op = +1, .sem_flg = 0 }, opDs0 = { 0, -1, 0 };
    if( semop(semid, &opAs0, 1) < 0 ){ … }
    // …
    if( semop(semid, &opDs0, 1) < 0 ){ … }
    // …
    return 0;
}
```

# POSIX Semaphores

- Более современный способ работы с семафорами. Альтернатива System V Semaphores. В рамках нашего курса НЕ ИСПОЛЬЗУЕМ.

- Пространство имён семафоров = имена из файловой системы
- **sem_open(“/myname”, ...)**: создать или открыть семафор по имени (спец. файл обычно /dev/shm/sem.myname на файловой системе tmpfs).
- **sem_post(...)**: увеличить семафор на 1.
- **sem_wait(...)**: пытаться уменьшить семафор на 1, ждать если в итоге получилось бы отрицательное значение.
- **sem_unlink()**: удалить семафор (спец. файл).