

Самарский национальный исследовательский университет им. акад. С.П. Королева

ОПЕРАЦИОННЫЕ СИСТЕМЫ СЕМЕЙСТВА UNIX

Часть III

Составитель: к.т.н., доц. К.Е. Климентьев

Самара 2015

1. Потоки. Моделирование средствами процессов

Потоки (нити, треды) – программы, разделяющие общее адресное пространство (т.е. код, данные, стек и т.п.). Достоинства: быстрота переключения, простота передачи данных.

Функция **clone()** создает точную копию текущего процесса, включая:

- CLONE_VM – адресное пространство;
- CLONE_FS – файловую систему;
- CLONE_FILES – дескрипторы файлов;
- CLONE_SIGHAND – обработчики сигналов;
- CLONE_PID – идентификатор процесса (**этого делать не надо!!!**).

```
#include <sched.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

/*Глобальные данные*/
char stack[10000], i, j;

/*Код и данные потока*/
int mythread(void * thread_arg) {
    for(int i=0;i<10;i++){ sleep(1); printf("\nMythread: i=%d",i);}
    printf("Thread1 finished\n");
}

/*Код и данные «главного» потока*/
int main() {
    clone(mythread, (void*)(stack+10000-1), CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND, NULL);
    for(j=0;j<10;j++){ sleep(1); printf("Main thread: j=%d\n",j); }
}
```

2. POSIX Threads - фреймворк pthreads

Основные функции:

- `pthread_create()` – создать поток;
- `pthread_exit()` – завершить работу потока изнутри;
- `pthread_cancel()` – завершить работу потока снаружи;
- `pthread_kill()` – послать сигнал потоку;
- `pthread_join()` – ждать завершения потока;
- `pthread_detach()` – отказаться от сохранения служебной информации о потоке.

```
/*Пример с общим адресным пространством и кодом, компиляция: gcc primer.c -lpthreads */
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

void *func(void *arg) {
    int loc_id = * (int *) arg;
    while (1) {
        printf("Thread %i is running\n", loc_id); sleep(1);
    }
}

int main() {
    int id1, id2, result;
    pthread_t thread1, thread2;
    id1 = 1; result = pthread_create(&thread1, NULL, func, &id1); pthread_detach(thread1);
    id2 = 2; result = pthread_create(&thread2, NULL, func, &id2); pthread_detach(thread2);
}
```

3. POSIX Threads (продолжение)

```
/*Пример с разным кодом*/
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

void *func1() { /* Поток 1 */
    int i; for (i=0;i<10;i++) { printf("Thread 1 is running\n"); sleep(1); }
}

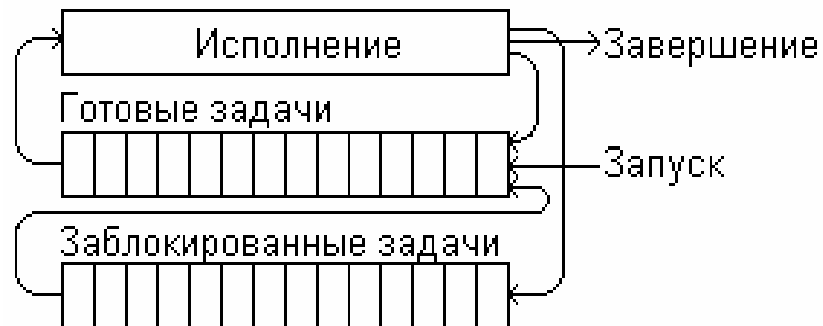
void *func2() { /* Поток 2 */
    int i; for (i=0;i<10;i++) { printf("Thread 2 is running\n"); sleep(1); }
}

int main() {
    int result, status1, status2;
    pthread_t thread1, thread2;
    result = pthread_create(&thread1, NULL, func, NULL);
    result = pthread_create(&thread2, NULL, func, NULL);
    pthread_join(thread1, &status1);
    pthread_join(thread2, &status2);
    printf("\nПотоки завершены с %d и %d", status1, status2); // Например, PTHREAD_CANCELLED
}
```

Статусы потоков сохраняются после завершения. Если выполнить `pthread_detach()`, то они будут недоступны.

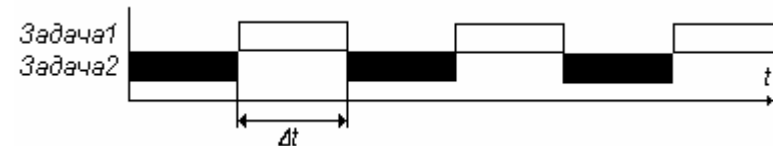
4. Организация параллельного выполнения задач

4.1. Граф многозадачности

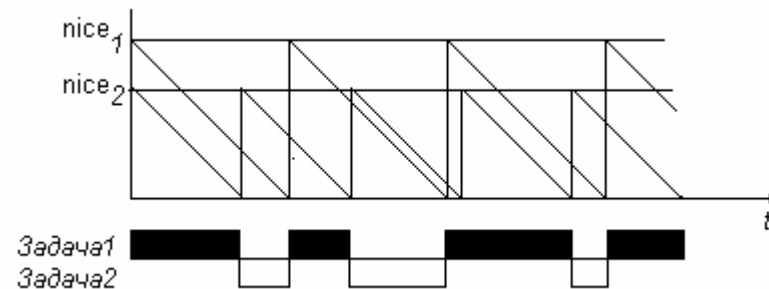


4.2. Алгоритмы диспетчеризации в UNIX

4.2.1. SCHED_RR (карусель) и SCHED_FIFO (очередь) – в pthreads не реализованы.

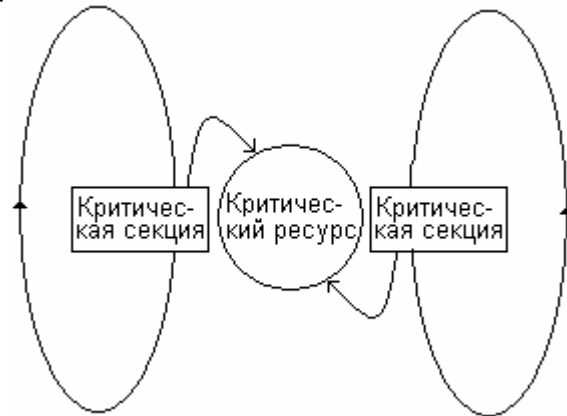


4.2.2. SCHED_OTHER (старение)

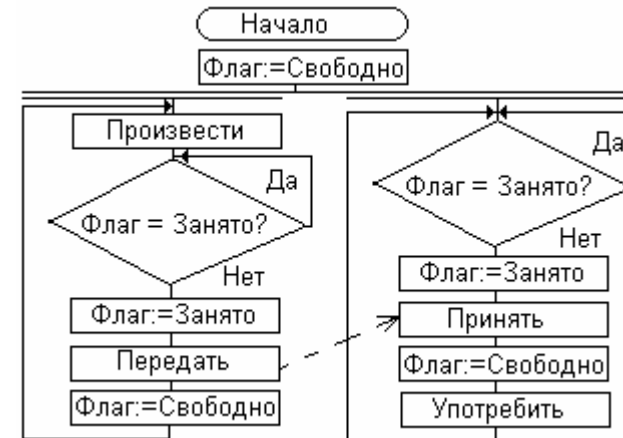


5. Проблема синхронизации задач

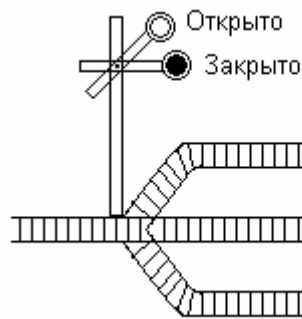
5.1. Проблема доступа к общему ресурсу



5.2. Решение при помощи блокирующего флага

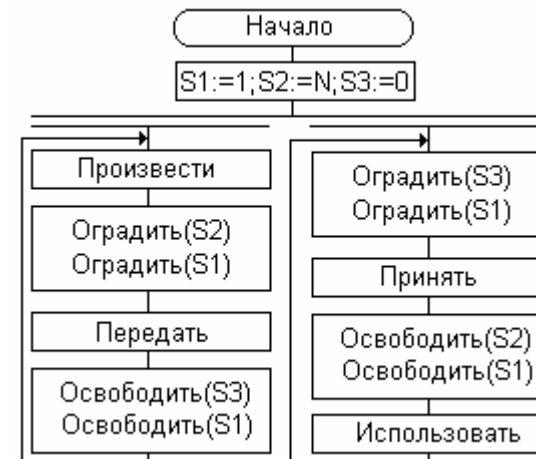


5.3. Принцип работы и определение семафора



1. Целая переменная S
2. $P(S)$ - операция «Оградить»
 - $S := S - 1$
 - Если $S < 0$, то текущая задача встает в очередь
3. $V(S)$ - операция «Освободить»
 - $S := S + 1$
 - Если $S > 0$, то 1-я в очереди задача продолжает работу

5.4. Решение при помощи семафоров



6. Семафоры

Основные функции:

- `sem_open()` – открыть именованный семафор
- `sem_close()` – закрыть именованный семафор
- `sem_unlink()` – удалить именованный семафор
- `sem_init()` – инициализировать неименованный семафор
- `sem_destroy()` – удалить неименованный семафор
- `sem_wait()` – если $S > 0$ то $S := S - 1$ и продолжить; иначе встать в очередь.
- `sem_trywait()` – возвращается с неудачей, если $S > 0$, то $S := S - 1$
- `sem_post()` – если очередь пуста, то $S := S + 1$ и продолжить; иначе первый в очереди продолжает работу

```
sem_t empty, full;
int data;
```

```
int main(){
    sem_init(&empty, SHARED, 1); // Установлен в 1
    sem_init(&full, SHARED, 0);  // Установлен в 0
    ...
}

void *Producer(void *arg){ // Сначала empty=1 full=0
    while(1) {
        sem_wait(&empty);    // empty:=empty-1
        data=Expression();
        sem_post(&full);     // full:=full+1
    }
}

void *Consumer(void *arg){ // Сначала empty=1 full=0
    while(1) {
        sem_wait(&full);     // full:=full-1
        Use(data);
        sem_post(&empty);    // empty:=empty+1
    }
}
```

7. Мьютексы (мутексы)

Мьютекс – двоичный семафор, принимающий значения $S=0$ или $S=1$. Основные функции:

- `init_mutex_init()` – инициализирует мьютекс
- `pthread_mutex_lock()` – попытка захвата мьютекса (при неудаче ждать)
- `pthread_mutex_trylock()` – попытка захвата мьютекса (при неудаче вернуть ошибку)
- `pthread_mutex_unlock()` – освобождение мьютекса
- `pthread_mutex_destroy()` – удаляет мьютекс

```
int data; pthread_mutex_t mutex;

void *Producer() {
    while(1) {
        pthread_mutex_lock(&mutex);
        printf("\nPing: %d", ++data); sleep(1);
        pthread_mutex_unlock(&mutex);
    }
}

void *Consumer() {
    while(1) {
        pthread_mutex_lock(&mutex);
        printf("\nPong: %d", --data); sleep(1);
        pthread_mutex_unlock(&mutex);
    }
}

int main() {
    pthread_t thread1, thread2;
    pthread_mutex_init(&mutex, NULL); data = 0;
    pthread_create(&thread1, NULL, &Producer, NULL);
    pthread_create(&thread2, NULL, &Consumer, NULL);
    sleep(60);
    pthread_cancel(thread1); pthread_cancel(thread2);
}
```


8. Условные переменные

Условные переменные – расширение мьютексов. Основные функции:

- `pthread_cond_init(&cond, NULL)` - инициализация
- `pthread_cond_wait(&cond, &mutex)` - разблокировать mutex + ждать + дожждаться + заблокировать mutex
- `pthread_cond_signal(&cond)` - послать сигнал разблокировки первому в очереди
- `pthread_cond_broadcast(&cond)` - послать сигнал всем

```
int data;
pthread_mutex_t mutex;
pthread_cond_t cond;

void *Producer() {
    while(1) {
        pthread_mutex_lock(&mutex);
        printf("\nPing: %d", ++data);
        pthread_cond_signal(&cond);
        sleep(1);
        pthread_mutex_unlock(&mutex);
    }
}
```

```
void *Consumer() {
    while(1) {
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond, &mutex); // Чтобы не делать холостой цикл, если условие
не наступило
        printf("\nPong: %d", --data);
        pthread_mutex_unlock(&mutex);
    }
}
```

```
int main() {
    pthread_t thread1, thread2;
    pthread_cond_init(&cond, NULL);
    pthread_mutex_init(&mutex, NULL);
    data = 0;
    pthread_create(&thread1, NULL, &Producer,
    NULL);
    pthread_create(&thread2, NULL, &Consumer,
    NULL);
    ...
}
```