Лекция 11

- Мьютекс и семафор
- Взаимная блокировка
- Задача “производитель-потребитель”

Мьютексы

```
acquire() {
  while (!available);
  available = false;
}
```

```
release() {
 available = true;
}
```

```
while (true) {
    acquire lock
       Критическая область
    release lock
       Некритическая область
}
```

Вызовы acquire и release выполняются **атомарно**!

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

pthread_mutex_t Mutex;//=PTHREAD_MUTEX_INITIALIZER;

char sh[6];
void* Thread( void* pParams );

int main( void ) {
 pthread_t thread_id;

 pthread_mutex_init(&Mutex, NULL);
 pthread_create(&thread_id, NULL, &Thread, NULL);
```

```
 while( 1 ){
  pthread_mutex_lock(&Mutex);
  printf("%s\n",sh);
  pthread_mutex_unlock(&Mutex);
}
return 0;
}
```

```
void* Thread( void* pParams ){
int counter = 0;
 while ( 1 ){
  pthread_mutex_lock(&Mutex);
  if(counter%2){
   sh[0]='H'; sh[1]='e'; sh[2]='l';
   sh[3]='l'; sh[4]='o'; sh[5]='\0';
  }
  else{
   sh[0]='B';sh[1]='y';sh[2]='e';
   sh[3]='_';sh[4]='u';sh[5]='\0';
  }
  pthread_mutex_unlock(&Mutex);
  counter++;
 }
}
```

Взаимоблокировка

```
pthread_mutex_lock(mutex1);
pthread_mutex_lock(mutex2);
Критическая область
```

```
pthread_mutex_lock(mutex2);
pthread_mutex_lock(mutex1);
Критическая область
```

```
pthread_mutex_lock(mutex);
pthread_mutex_lock(mutex);
Критическая область
```

```
int main( void ) {
 pthread_t thread_id;
 pthread_mutexattr_t  attr;

 pthread_mutexattr_init(&attr);
 pthread_mutexattr_settype(&attr,PTHREAD_MUTEX_RECURSIVE);
                                 //PTHREAD_MUTEX_ERRORCHECK);

 pthread_mutex_init(&Mutex, &attr);
 pthread_mutexattr_destroy(&attr);

 pthread_create(&thread_id, NULL, &Thread, NULL);
```

```
 while( 1 ){
  pthread_mutex_lock(&Mutex);
  pthread_mutex_lock(&Mutex);//взаимоблокировка(deadlock)
  printf("%s\n",sh);
  pthread_mutex_unlock(&Mutex);
  pthread_mutex_unlock(&Mutex);
 }
 return 0;
}
```

Семафоры

При изменении глобальной переменной S используются атомарные операции.

```
P(S) {
while (S <= 0);
S--;
}
```

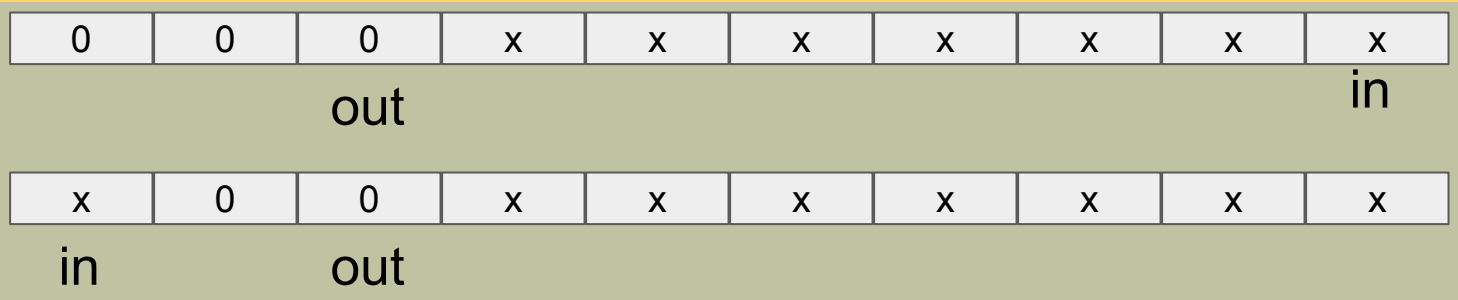
```
V(S) {
  S++;
}
```

```
#include <semaphore.h>
#include <unistd.h>
#include <stdio.h>

sem_t  Sem;

char sh[6];
void* Thread( void* pParams );

int main( void ) {
 int n=0;
 pthread_t thread_id;

 sem_init(&Sem, 0, 1); // анонимный семафор
```

```
pthread_create(&thread_id, NULL, &Thread, NULL);

 while(n<100 ){
  sem_wait(&Sem);
  printf("%s\n",sh);
  sem_post(&Sem);
  usleep(1000);
  n++;
 }
 pthread_join(thread_id, NULL);

 sem_destroy(&Sem);
 return 0;
}
```

```
void* Thread( void* pParams ){
int counter = 0;

while ( counter<1000 ){
 sem_wait(&Sem);
 if(counter%2){
   sh[0]='H';sh[1]='e';sh[2]='l';
   sh[3]='l';sh[4]='o';sh[5]='\0';
 }
 else{
   sh[0]='B';sh[1]='y';sh[2]='e';
   sh[3]='_';sh[4]='u';sh[5]='\0';
 }
 sem_post(&Sem);
 usleep(500);
 counter++; }}
```

Задача “производитель-потребитель”




```
while (true) {
  while (count == BUFFER SIZE);
  buffer[in] = next produced;
  in = (in + 1) % BUFFER SIZE;
  count++;
}
```

```
while (true) {
  while (count == 0);
  next consumed = buffer[out];
  out = (out + 1) % BUFFER SIZE;
  count--;
}
```

```
#include <malloc.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

struct portion{
struct portion* next;
};
struct portion *buffer=NULL;

pthread_mutex_t Mutex=PTHREAD_MUTEX_INITIALIZER;

sem_t buffer_size;
```

```
void* consumer(void* params){
 while(1){
  struct portion *next_portion;
  sem_wait(&buffer_size);
  pthread_mutex_lock(&Mutex);
   printf("%X\n", buffer);
   next_portion=buffer;
   buffer=buffer->next;
  pthread_mutex_unlock(&Mutex);
 // manipulate(next_portion);
  free(next_portion);
}
return 0;
}
```

```
void* producer(void* params){
 while(1){
  if(i++>200) break;
  usleep(100);
  struct portion *new_portion;
  new_portion=(struct portion*)calloc(1,
                                 sizeof(struct portion));
  pthread_mutex_lock(&Mutex);
   new_portion->next=buffer;
   buffer=new_portion;
   sem_post(&buffer_size);
  pthread_mutex_unlock(&Mutex);
};
return 0;
}
```

```
int main(){
 int i;
 sem_init(&buffer_size, 0, 0);
 pthread_t producer_id, consumer_id[3];

pthread create(&producer_id, NULL, &producer, NULL);
 for(i=0;i<3;i++)
  pthread_create(&consumer_id[i], NULL, &consumer, NULL);

 pthread join(producer_id, NULL);
 for(i=0;i<3;i++)
  pthread_join(consumer_id[i], NULL);

 return 0;
}
```