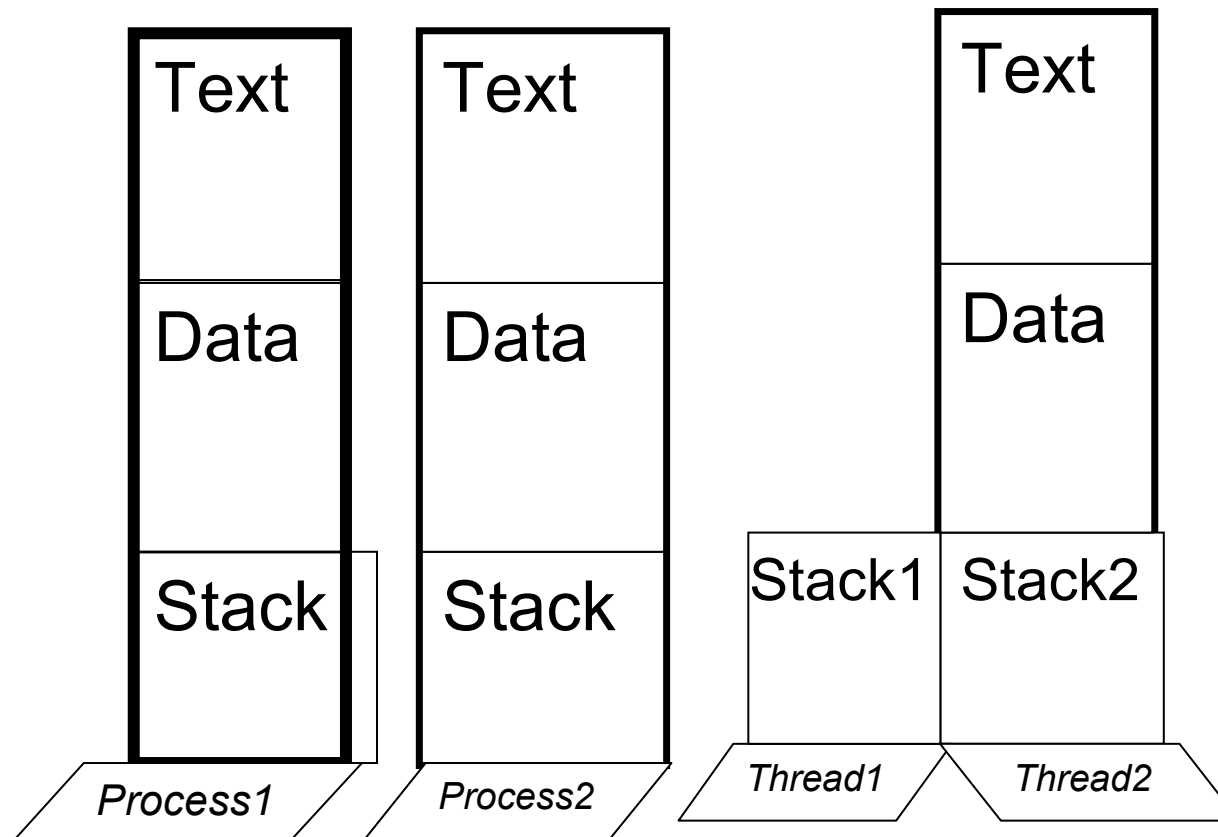


Threads (POSIX threads, pthreads)

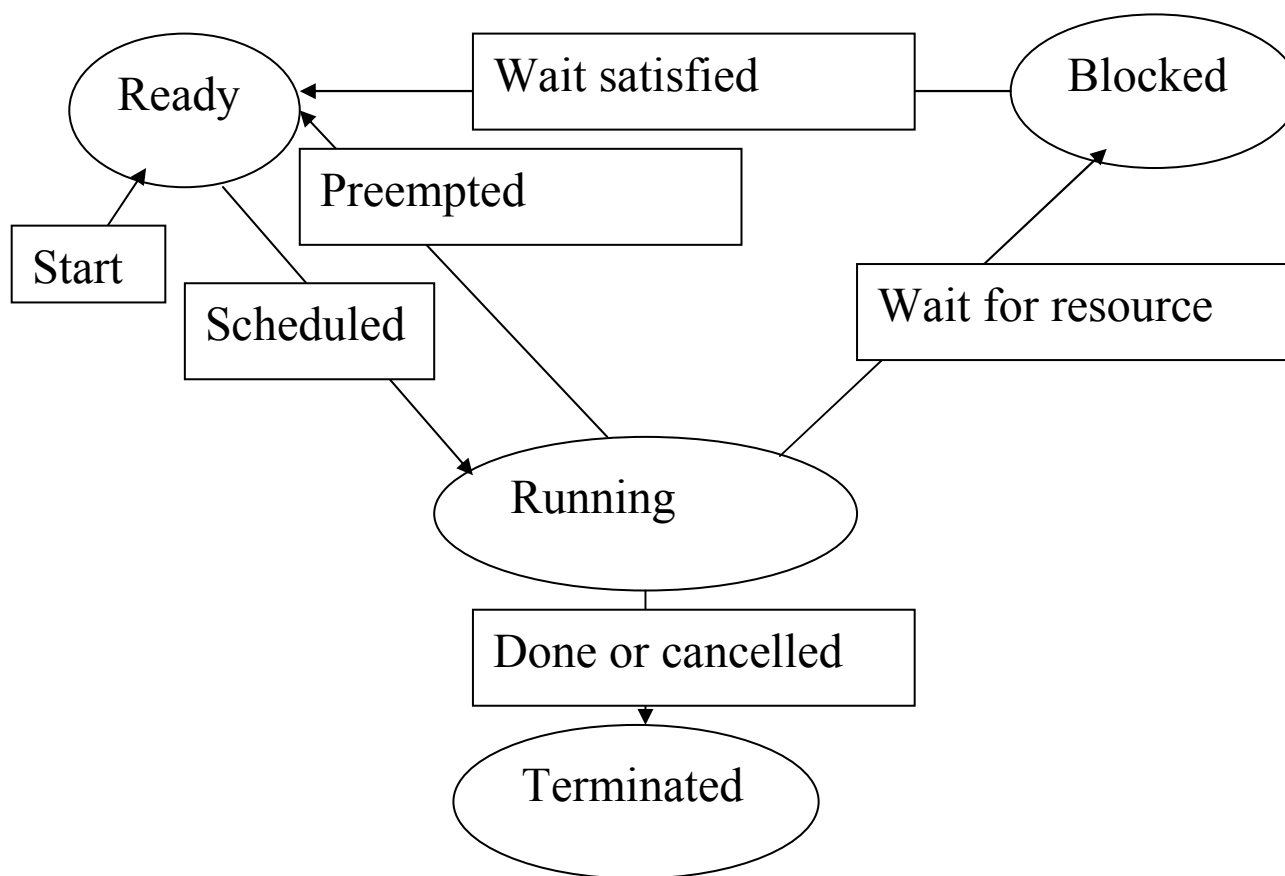
Proces: to, čo operačný systém robí s programom

Thread: tok riadenia v procese (s vlastným stackom)



Threads (POSIX threads, pthreads)

Stavový diagram threadu sa podobá na stavový diagram procesu



Threads (POSIX threads, pthreads)

API

pthread_create()

pthread_equal()

pthread_join()

pthread_self()

pthread_mutex_init()

pthread_mutex_destroy()

pthread_mutex_lock()

pthread_mutex_unlock()

pthread_cond_init()

pthread_cond_destroy()

pthread_cond_signal()

pthread_cond_wait()

Threads (POSIX threads, pthreads)

Príklad: test.c

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

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

```
void *thread_routine(void* arg) {  
    printf("Inside newly created thread\n");  
}
```

```
int main() {  
    pthread_t thread_id;  
    void *thread_result;  
    pthread_create( &thread_id, NULL, thread_routine, NULL );  
    printf("Inside main thread \n");  
    pthread_join(thread_id, &thread_result );  
}
```

```
gcc test.c -lpthread
```

Threads (POSIX threads, pthreads)

Predošlý program netestuje návratové hodnoty funkcií `pthread_create()` a `pthread_join()`. Na slajde to robiť netreba, v kóde áno:

```
if ((pthread_create(&thread_id, NULL, thread_routine, NULL) != 0)
{
    ERROR("pthread_create() failed\n");
}
```

VIth Commandment

If a function be advertised to return an error code in the event of difficulties, thou shalt check that code, yea, even though the checks triple the size of thy code and produce aches in thy typing fingers, for if thou thinkest „it cannot happen to me“, the gods shall surely punish thee for thy arrogance.

Ten Commandments for C programmers (Henry Spencer)

1. Thou shalt run lint frequently and study its pronouncements with care, for verily its perception and judgement oft exceed thine.
2. Thou shalt not follow the NULL pointer, for chaos and madness await thee at its end.
3. Thou shalt cast all function arguments to the expected type if they are not of that type already, even when thou art convinced that this is unnecessary, lest they take cruel vengeance upon thee when thou least expect it.
4. If thy header files fail to declare the return types of thy library functions, thou shalt declare them thyself with the most meticulous care, lest grievous harm befall thy program.
5. Thou shalt check the array bounds of all strings (indeed, all arrays), for surely where thou typest "foo" someone someday shall type "supercalifragilisticexpialidocious".
6. If a function be advertised to return an error code in the event of difficulties, thou shalt check for that code, yea, even though the checks triple the size of thy code and produce aches in thy typing fingers, for if thou thinkest "it cannot happen to me", the gods shall surely punish thee for thy arrogance.
7. Thou shalt study thy libraries and strive not to re-invent them without cause, that thy code may be short and readable and thy days pleasant and productive.
8. Thou shalt make thy program's purpose and structure clear to thy fellow man by using the One True Brace Style, even if thou likest it not, for thy creativity is better used in solving problems than in creating beautiful new impediments to understanding.
9. Thy external identifiers shall be unique in the first six characters, though this harsh discipline be irksome and the years of its necessity stretch before thee seemingly without end, lest thou tear thy hair out and go mad on that fateful day when thou desirest to make thy program run on an old system.
10. Thou shalt foreswear, renounce, and abjure the vile heresy which claimeth that "All the world's a VAX", and have no commerce with the benighted heathens who cling to this barbarous belief, that the days of thy program may be long even though the days of thy current machine be short.

Pthreads: producer-consumer

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int shared_data =1;
void *consumer(void* arg) {
    int i;
    for (i =0; i < 30; i ++ ){
        pthread_mutex_lock( &mutex );
        shared_data --; /* Critical Section. */
        pthread_mutex_unlock( &mutex );
    }
    printf("Returning from Consumer =%d\n", shared_data);
}

void main() {
    int i;
    pthread_t thread_id;
    pthread_create( & thread_id, NULL, consumer, NULL );
    for(int i =0; i < 30; i ++ ){
        pthread_mutex_lock( &mutex );
        shared_data ++; /* Producer Critical Section. */
        pthread_mutex_unlock( &mutex );
    }
    printf("End of main =%d\n", shared_data);
}
```

Pthreads: producer-consumer (bounded buffer)

```
#define QUEUE_SIZE 10
#define ITERATIONS 1000

int in = 0, out = 0;
int shared_data = 1;
int n_consumer = 0;

pthread_mutex_t read_mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t write_mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t qempty_cond_mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t q_notempty_cond =PTHREAD_COND_INITIALIZER;
pthread_mutex_t qfull_cond_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t q_notfull_cond = PTHREAD_COND_INITIALIZER;

void main() {
    pthread_t thread_consumer;
    pthread_create(&thread_consumer, NULL, consumer, NULL);
    producer();
    pthread_join(&thread_consumer);
}
```

Pthreads: producer-consumer (bounded buffer)

```
void *consumer(void* arg) {
    int i;
    n_consumer ++;
    for (i =0; i < ITERATIONS; i++) {
        pthread_mutex_lock( &mutex );

        while ( queue_is_empty() ){
            pthread_cond_wait(&q_notempty_cond, &mutex );
        }
        /* read from queue[ in ] */
        in = (in +1) % QUEUE_SIZE;

        pthread_cond_signal(&q_notfull_cond);
        pthread_mutex_unlock( &mutex );

    }
    printf("Returning from Consumer\n");
    n_consumer --;
}
```

Pthreads: producer-consumer (bounded buffer)

```
void *producer(void* arg ) {
    int i;
    for (i =0; n_consumer; i ++ ) {
        pthread_mutex_lock( &mutex );

        while ( queue_is_full() ){
            pthread_cond_wait(&q_notfull_cond, &mutex );
        }

        /* write to queue[out] */
        out = (out +1) % QUEUE_SIZE;
        pthread_mutex_unlock( &mutex );
        pthread_cond_signal(&q_notempty_cond );
    }
    printf("Returning from Producer\n");
}
```

Pthreads: rules of thumb

- Shared data should always be accessed through a single mutex
- Think of a boolean condition (expressed in terms of program variables) for each condition variable. Every time the value of the boolean condition may have changed, `signal()` the condition variable
 - Call `Signal` with a locked mutex; and only call it when you are certain that another thread is waiting for the signal (unlike in previous examples... oh well)
- Globally order locks, acquire in order in all threads

Pthreads: Linux (contention scope)

- Contention scope is the POSIX term for describing bound and unbound threads
 - A bound thread is said to have system contention scope, i.e. it contends with all threads in the system
 - An unbound thread has process contention scope, i.e. it contends with threads in the same process
- Solaris LWP switching is cheap, Linux is one-to-one anyways...