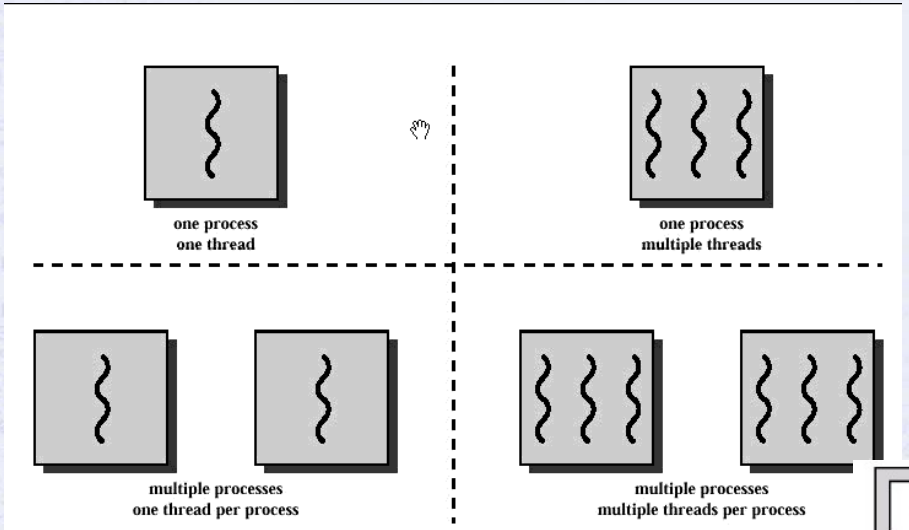
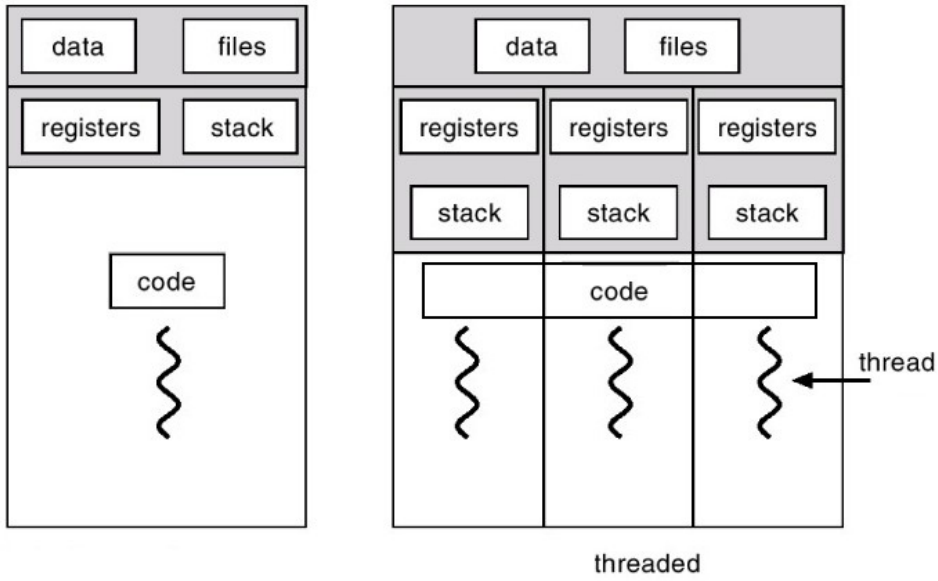
EPL372
Lab Exercise 2:
Threads and pThreads

Εργαστήριο 2

Πέτρος Παναγή

Threads Vs Processes

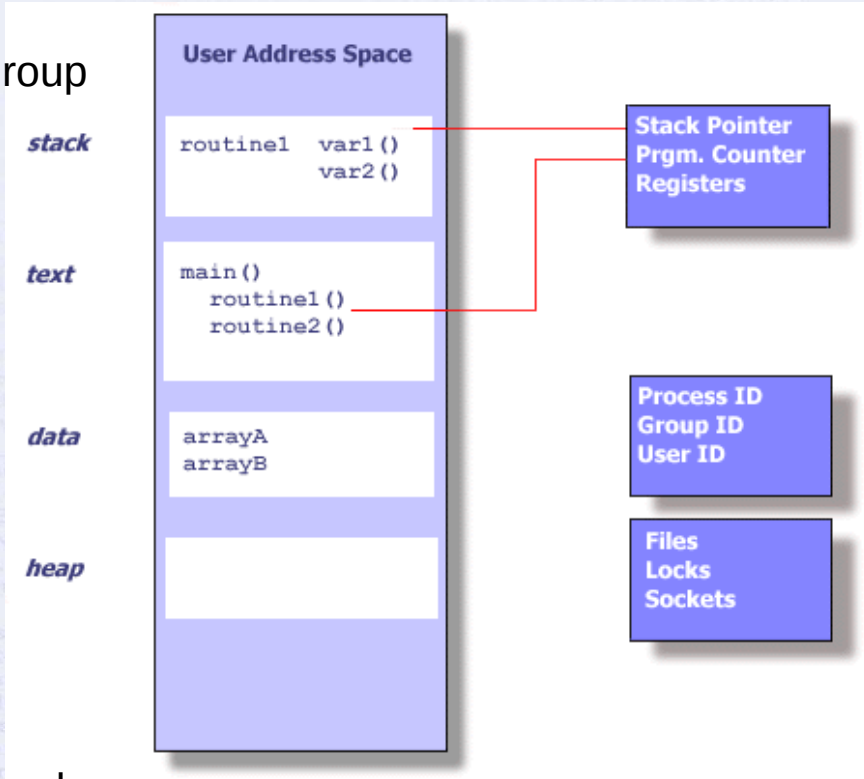




Process

A process is created by the operating system, and requires a fair amount of "overhead". Processes contain information about program resources and program execution state, including:

- Process ID, process group ID, user ID, and group ID
- Environment
- Working directory.
- Program instructions
- Registers
- Stack
- Heap
- File descriptors
- Signal actions
- Shared libraries
- Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory).

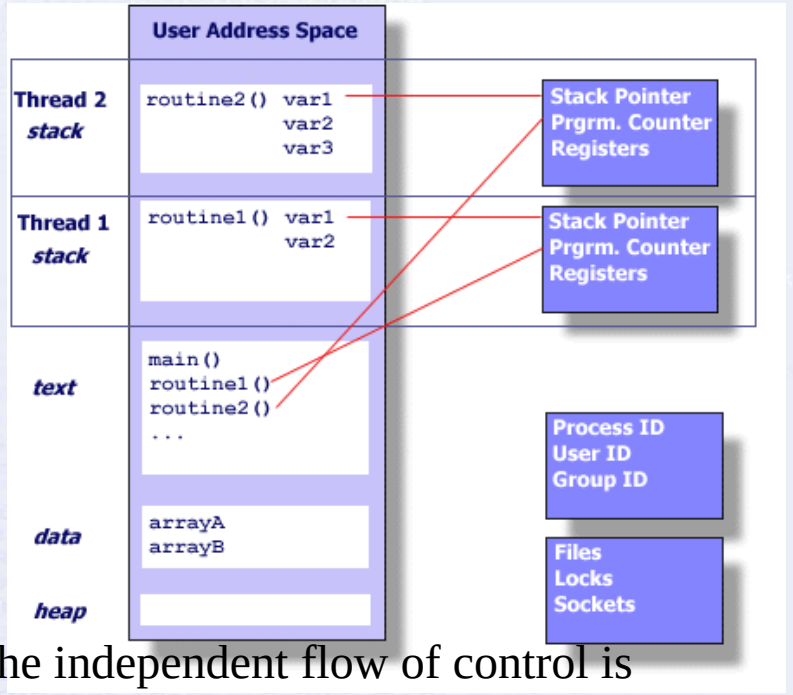


Threads characteristics

Threads use and exist within these process resources, yet are able to be scheduled by the operating system and run as independent entities largely because they duplicate only the bare essential resources that enable them to exist as executable code.

Most operating systems support programs that have multiple threads of execution. Although implementations differ, they usually possess the following common characteristics:

- **Shared address space** - threads can read/write the same variables and execute the same code.
- **Private execution context** - every thread has its own set of registers
- **Private execution stack** - every thread has address space reserved for its stack
- **Thread - process association** - threads exist within and use the resources of a process. They cannot exist outside of a process



The independent flow of control is accomplished because a thread maintains its own:

- Stack pointer
- Registers
- Scheduling properties (such as policy or priority)
- Set of pending and blocked signals
- Thread specific data.

https://computing.llnl.gov/tutorials/pthreads/

In Summary

- In summary, in the UNIX environment a thread:
  - Exists within a process and uses the process resources
  - Has its own independent flow of control as long as its parent process exists and the OS supports it
  - Duplicates only the essential resources it needs to be independently schedulable
  - May share the process resources with other threads that act equally independently (and dependently)
  - Dies if the parent process dies - or something similar
  - Is "lightweight" because most of the overhead has already been accomplished through the creation of its process.
- Because threads within the same process share resources:
  - Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads.
  - Two pointers having the same value point to the same data.
  - Reading and writing to the same memory locations is possible, and therefore requires explicit *synchronization by the programmer*. (This is the big Problem with threads)

Linux POSIX threads library **pthread_create**

pthread_create - thread creation

SYNOPSIS

#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);

The pthread_create() function is used to create a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes are used. If the attributes specified by attr are modified later, the thread's attributes are not affected. Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread.

http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html

http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_create.html

pthread_join - wait for thread termination

SYNOPSIS

#include <pthread.h>

int pthread_join(pthread_t *thread*, void ***value_ptr*);

DESCRIPTION

The ***pthread_join*()** function shall **suspend execution of the calling thread** until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join*() call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit()* by the terminating thread shall be made available in the location referenced by *value_ptr*. When a *pthread_join*() returns successfully, the target thread has been terminated.

pthread_create: Example 1

```c
/* Compile Using:
gcc -lpthread -Werror -Wall thread-create.c -o thread-create
*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>     /* srand, rand */
#include <unistd.h>

int randomNumber;

void* childThread (void* threadID)
{
    sleep(10-(long)threadID);
    printf ("Child Thread %ld is called with RANDOM %d\n", (long)threadID,randomNumber++);
    return NULL;
}
/* The main program. */
int main ()
{
    pthread_t thread_id;
    long threadCount;
    srand(time(NULL));
    randomNumber = rand()%1000;
    /* Create a new thread. The new thread will run the print_xs  function. */
    for (threadCount = 1; threadCount< 10; threadCount++){
        pthread_create (&thread_id, NULL, &childThread, (void *)threadCount);
        printf ("Parent has created thread: %u \n", (unsigned int)thread_id);
        }
    sleep(10);
    return 0;
}
```

gcc -lpthread -Werror -Wall thread-create.c -o thread-create

pthread_create: Example 2

Download and Run the Second Example:
Do the Changes in the code and explain why the results differ.