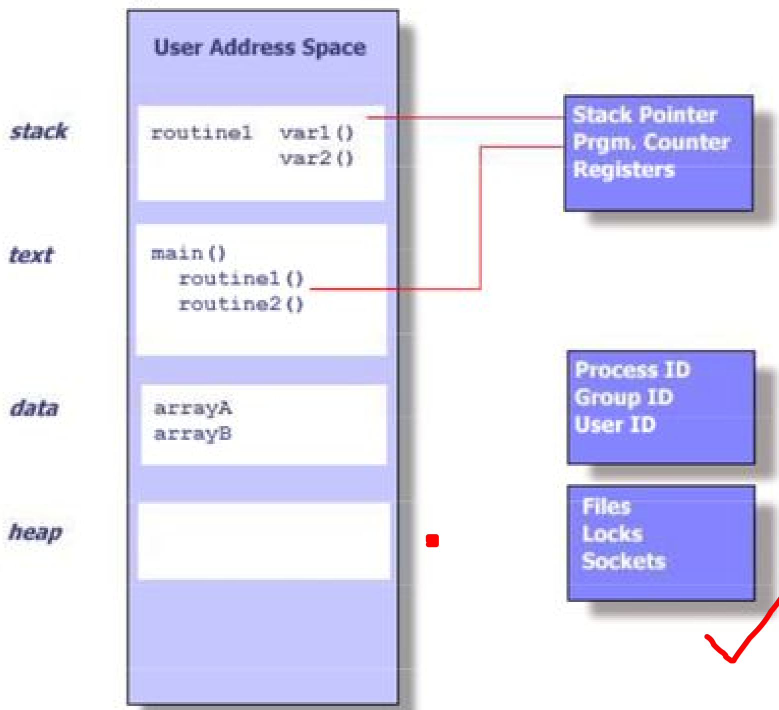
Tutorial 4

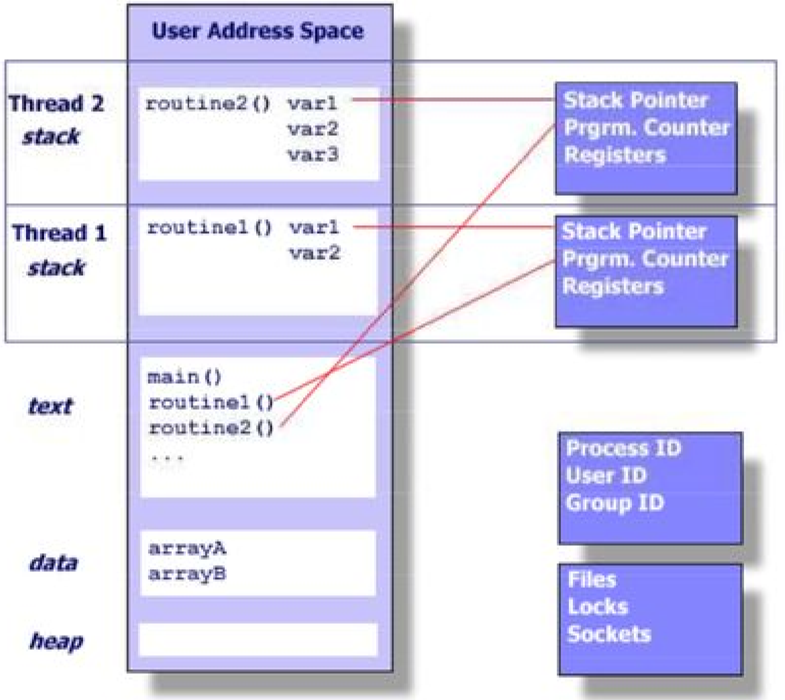
Operating System Lab

Thread

- A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*.
- Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals.
- But, like process, a thread has its own program counter (PC), a register set, and a stack space.



UNIX PROCESS



THREADS WITHIN A UNIX PROCESS

Thread

- All threads within a process share the same address space.
- Threads in the same process share:
  - Process instructions
  - Most data
  - open files (descriptors)
  - signals and signal handlers
  - current working directory
  - User and group id

Thread

- Each thread has a unique:
  - Thread ID
  - set of registers, stack pointer
  - stack for local variables, return addresses
  - signal mask
  - priority
  - Return value: errno
- Thread operations include thread creation, termination, synchronization (joins,blocking), scheduling, data management and process interaction.

pthread(POSIX threads)

- The primary motivation for using Pthreads is to realize potential program performance gains.
- Attributes of pthread
    - pthread_create(thread,attr,start_routine,arg)
    - pthread_exit(status)
    - pthread_attr_init(attr)
    - pthread_attr_destroy(attr)

Creating threads

- Initially, your main() program comprises a single, default thread. All other threads must be explicitly created by the programmer.
- pthread_create creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.
- pthread_create arguments:
  - ✓ thread: An opaque, unique identifier for the new thread returned by the subroutine.
  - ✓ attr: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  - ✓ start_routine: the C routine that the thread will execute once it is created.
  - ✓ arg: A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

Thread Attributes

- By default, a thread is created with certain attributes. Some of these attributes can be changed by the programmer via the thread attribute object.
- ✓ *pthread_attr_init* and *pthread_attr_destroy* are used to initialize/destroy the thread attribute object.

Compile and Execute Threaded Programs

Compile:

```
gcc thread.c -o thread.o -lpthread
```

Execute:

```
./thread
```

Terminating threads

- There are several ways in which a pthread may be terminated:
  - ✓ The thread returns from its starting routine (the main routine for the initial thread).
  - ✓ The thread makes a call to the pthread_exit subroutine.
  - ✓ The thread is canceled by another thread via the pthread_cancel routine.
  - ✓ The entire process is terminated due to a call to either the exec or exit subroutines.
- *pthread_exit* is used to explicitly exit a thread. Typically, the *pthread_exit*() routine is called after a thread has completed its work and is no longer required to exist.
- If main() finishes before the threads it has created, and exits with *pthread_exit*(), the other threads will continue to execute. Otherwise, they will be automatically terminated when main() finishes
- Cleanup: the *pthread_exit*() routine does not close files; any files opened inside the thread will remain open after the thread is terminated.

Passing arguments to threads

- The `pthread_create()` routine permits the programmer to pass one argument to the thread start routine.
- For cases where multiple arguments must be passed, this limitation is easily overcome by creating a structure which contains all of the arguments, and then passing a pointer to that structure in the `pthread_create()`routine.
- All arguments must be passed by reference and cast to (void *).

Joining threads

- Routines:
  - pthread_join(threadid,status)
  - pthread_detach(threadid,status)
  - pthread_attr_setdetachstate(attr,detachstate)
  - pthread_attr_getdetachstate(attr,detachstate)
- "Joining" is one way to accomplish synchronization between threads.
- ✓ The pthread_join() subroutine blocks the calling thread until the specified threadid thread terminates.
- The programmer is able to obtain the target thread's termination return status if it was specified in the target thread's call to pthread_exit()

Detaching threads

- ✓ The `pthread_detach()` routine can be used to explicitly detach a thread even though it was created as joinable.
- There is no converse routine.