Threads

- A thread is a single sequence stream within a process. An independent stream of instructions that can be scheduled to run as such by the operating system.
- Also called lightweight processes.

Threads vs Processes

- Threads are not independent from each other.
- Threads share code section, data section and OS resources (open files/file handlers and signals) with other threads.
- Threads have their own program counter (PC), register set and a stack space.
- Processes can contain multiple threads but threads cannot contain multiple processes.

(see programs 5-threads.c and 5-processes.c)

Multithreading

- Threads offer a way to improve an application through parallelism.
- Threads operate faster than processes because:

1. Thread creation is much faster.
2. Context switching between threads is much faster.
3. Threads can be terminated easily.
4. Communication between threads is much faster.

POSIX Threads

- POSIX - Portable Operating System Interface UNIX
- Designed to work on any Unix system.
- Provide some API's (pthread API's) which can be used to do some basic things with threads (creating, deleting etc.)
- A single process can contain multiple threads, all of which are executing the same program.
- These threads share the same global memory (data and heap segments), but each thread has its own stack (automatic variables).

Functions

pthread_create()

```
#include <pthread.h>

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

- thread refers to the pointer to the thread that has to be created.
- pthread_attr_t refers to the thread attributes. Customization of threads.
- start_routine refers to the pointer to the function that has to be executed using the thread.
- void *arg refers to the arguments to be passed to the running function.
- The function returns 0 on SUCCESS and an error value/number on failure.

pthread_join()

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
```

- The pthread_join() function waits for the thread specified by thread to terminate.
- If that thread has already terminated, then pthread_join() returns immediately.
- it the retval is not NULL, then pthread_join() copies the exit status of the target thread into the location.

pthread_exit()

```
#include <pthread.h>

void pthread_exit(void *retval);
```

- The pthread_exit() function terminates the calling thread.
- This function can be called to return a value from a threaded function and then exit/return from it.
- Calling pthread_exit in main thread will result in the termination of main just like calling exit() would however the process would wait for the termination of all threads in this case.

- If we want to start some threads in main thread, leave them running and exit from main.

pthread_detach()

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

- Detaches a thread.
- When a detached thread terminates, its resources are automatically released back to the system without the need for another thread to join with the terminated thread.
- Attempting to detach an already detached thread results in unspecified behavior.
- Returns 0 on success, error number on error.
- A detached thread is not joinable i.e., we cannot call pthread_join() on it.
- Process will not finish execution until all of the detached threads finish their respective execution.

pthread_attr_setstacksize()

```
#include <pthread.h>

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);

int pthread_attr_getstacksize(const pthread_attr_t *restrict attr, size_t
*restrict stacksize);
```

- The pthread_attr_setstacksize() function sets the stack size

attribute of the thread attributes object referred to by attr to

the value specified in stacksize.

- The stack size attribute determines the minimum size (in bytes) that will be allocated for threads created using the thread

attributes object attr.

- The pthread_attr_getstacksize() function returns the stack size attribute of the thread attributes object referred to by attr in the buffer pointed to by stacksize.

- On success, these functions return 0; on error, they return a non-zero error number.

Race Condition

- Condition of a program where its behavious depends on relative timing ofr interleaving of multiple threads or processes.
- Two threads are basically racing to see which one of them gets to write first or write last.
- *Thread-safe* is the term used to describe a program, code, or data structure free of race conditions when accessed by multiple threads.
- Only occurs on a multi-core processor. Very unlikely to occur on single-core processors.
- **Fix**: Mutex Locks (Mutual Exclusion) - a computing abstraction that allows one thread to exclude other threads from the space it is working in.

Linux Documentation:

```
man pthreads
```

debugging tools for pthread ? how to use them? valgrinf helgrind strace, perf, gdb.

Q. What's the difference in process control block and thread control blocks?

A.

Process Control Block: A process control block (PCB) is a data structure used by computer operating systems to store all the information about a process. It is also known as a process descriptor. When a process is created (initialized or installed), the operating system creates a corresponding process control block.

This specifies the process state i.e. new, ready, running, waiting or terminated.

Thread Control Block: Thread Control Block (TCB) is a data structure in the operating system kernel which contains thread-specific information needed to manage it. The TCB is "the manifestation of a thread in an operating system."

References

- Programming with Threads - Jacob Sorber
- Unix Threads in C - CodeVault
- Operating System Lab (with Inter-Process Communication)
- Misellaneous
- Multithreading in C

- Race Condition
- Multithreading - Computerphile
- Stack sizes in threads
- Raw Linux Threads via System Calls
- Helgrind
- Helgrind 2
- Process Control Block
- Thread Control Block