14-UNIX

POSIX Threads

Chapter 29

Threads (1 of 2)

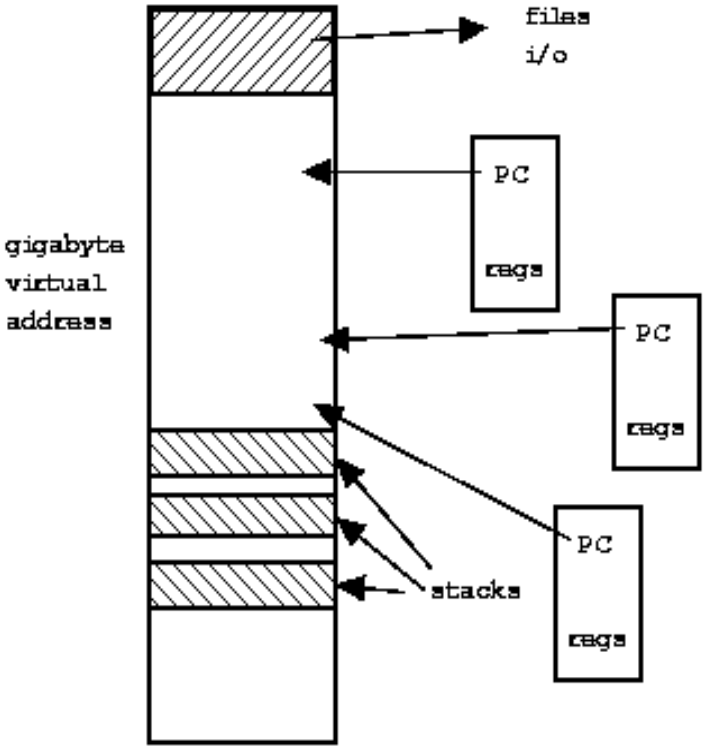
- Like processes, threads are mechanism that permit an application to perform MULTIPLE tasks concurrently (LPI – 29-1).
- Think of a thread as a “procedure” that runs independently from its main program.
- Multi-threaded programs are where several procedures are able to be scheduled to run simultaneously and/or independently by the OS.
- A Thread exists within a process and uses the process resources.

Threads (2 of 2)

- Threads only duplicate the essential resources they need to be independently schedulable.
- A thread will terminate if the parent process terminates.
- A thread is “lightweight” because most of the overhead has already been accomplished through the creation of the process.

Threads

- Each thread has its own stack, PC, registers
  - Share address space, files,..



From our book
LPI Page 618

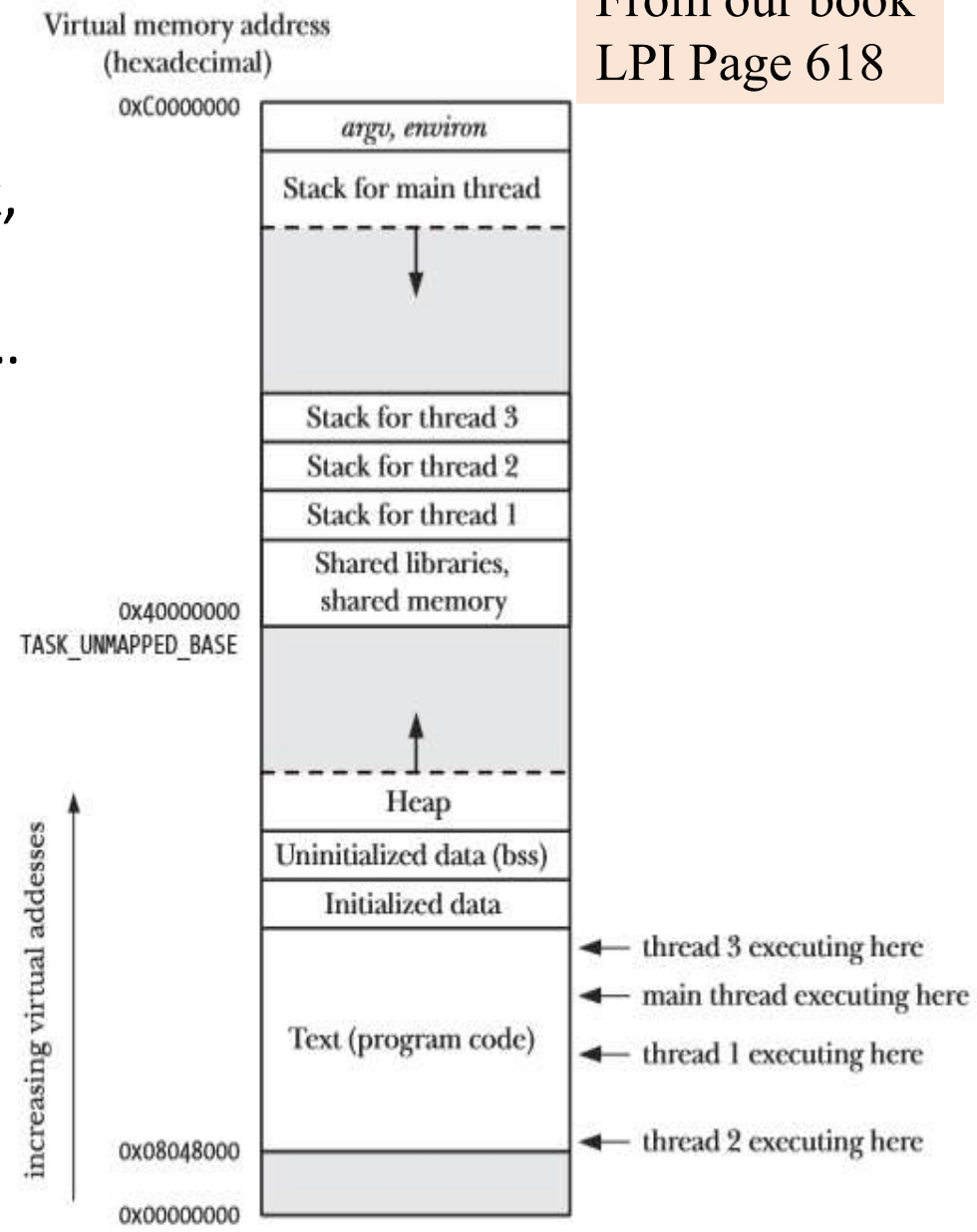


Figure 29-1: Four threads executing in a process (Linux/x86-32)

POSIX Threads (PThreads)

- For UNIX systems, implementations of threads that adhere to the IEEE POSIX 1003.1c standard are Pthreads.

- Pthreads are C language programming types defined in the pthread.h header/include file.

- How to compile:
  - gcc hello.c –lpthread –o hello

Why Use Pthreads

- The primary motivation behind Pthreads is improving program performance (**MAY** NOT BE A BENEFIT if your system has a single processor).

- Can be created with much less OS overhead.

- Needs fewer system resources to run.

Threads (clone) vs Forks (TLPI book)

Table 28-3: Time required to create 100,000 processes using *fork()*, *vfork()*, and *clone()*

Method of process creation	Total Virtual Memory					
	1.70 MB		2.70 MB		11.70 MB	
	Time (secs)	Rate	Time (secs)	Rate	Time (secs)	Rate
fork()	22.27 (7.99)	4544	26.38 (8.98)	4135	126.93 (52.55)	1276
vfork()	3.52 (2.49)	28955	3.55 (2.50)	28621	3.53 (2.51)	28810
clone()	2.97 (2.14)	34333	2.98 (2.13)	34217	2.93 (2.10)	34688
fork() + *exec()*	135.72 (12.39)	764	146.15 (16.69)	719	260.34 (61.86)	435
vfork() + *exec()*	107.36 (6.27)	969	107.81 (6.35)	964	107.97 (6.38)	960

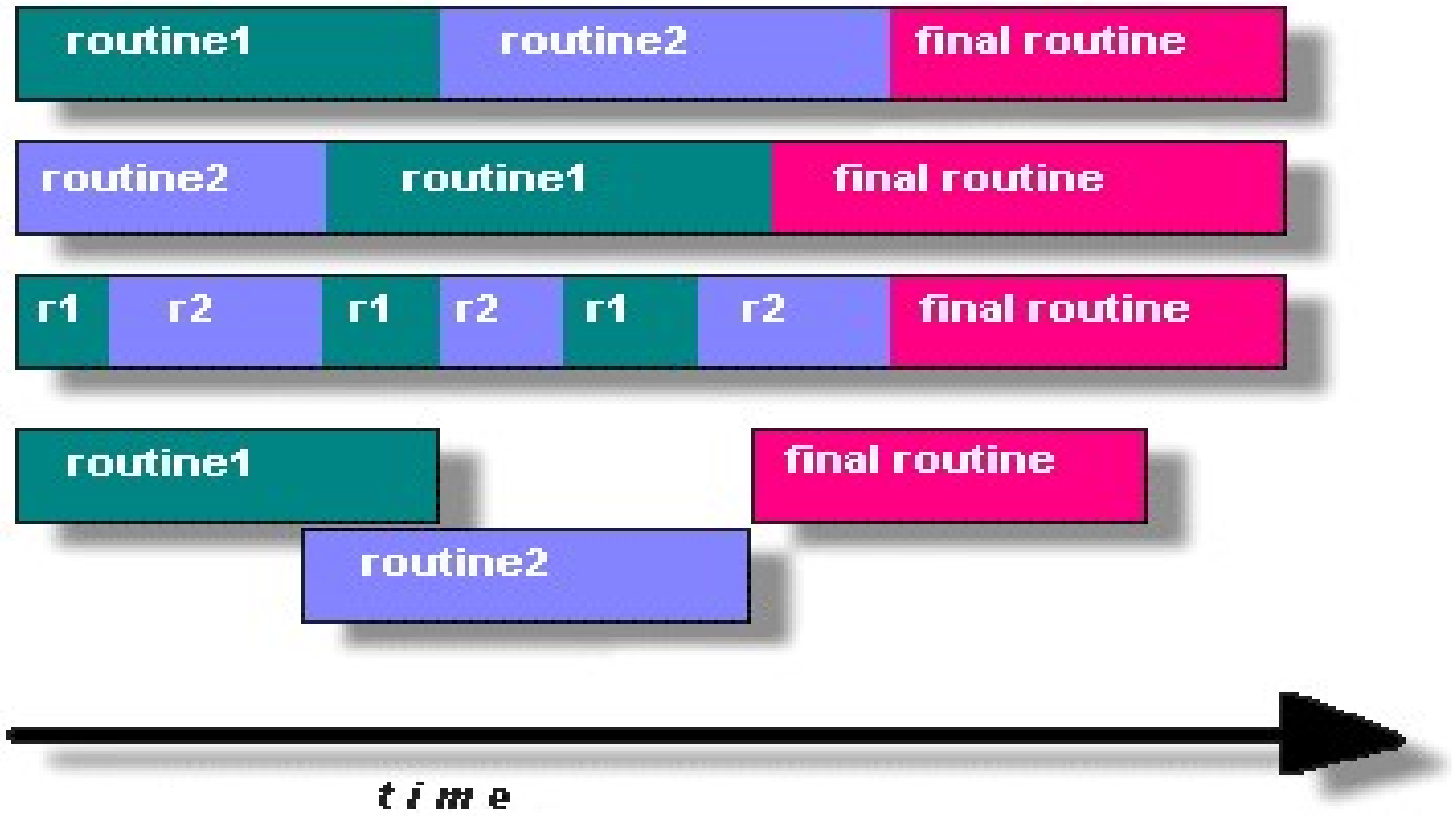
Thread creation is faster than process creation—typically, ten times faster or better. (On Linux, threads are implemented using the *clone()* system call, and Table 28-3, on page 610, shows the differences in speed between *fork()* and *clone()*.) Thread creation is faster because many of the attributes that must be duplicated in a child created by *fork()* are instead shared between threads. In particular, copy-on-write duplication of pages of memory is not required, nor is duplication of page tables.

Designing Pthreads Programs (1 0f 2)

- Pthreads are best used with programs that can be organized into discrete, independent tasks which can execute concurrently.

- Example: routine 1 and routine 2 can be interchanged, interleaved and/or overlapped in real time.

Candidates for Pthreads



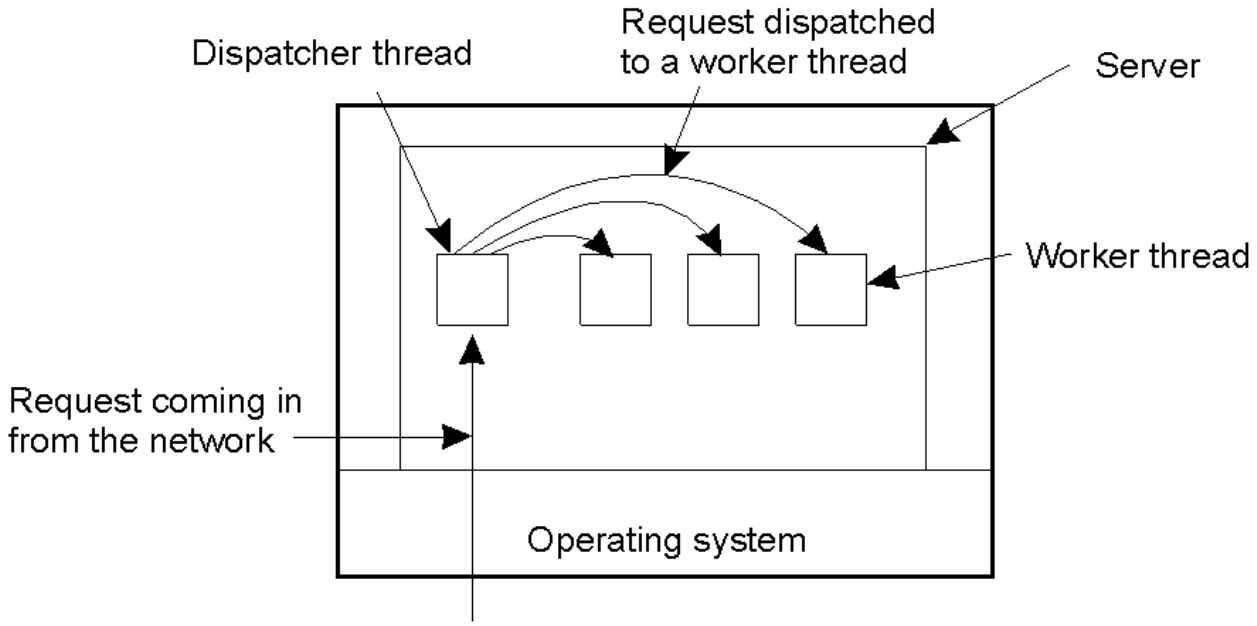
Designing Pthreads (2 of 2)

- Common models for threaded programs:
  - Manager/Worker: manager assigns work to other threads, the workers. Manager handles input and hands out the work to the other tasks.
  - Pipeline: task is broken into a series of sub-operations, each handled in series but concurrently, by a different thread.

Multi-threaded Server Example

Apache web server: pool of pre-spawned worker threads

- Dispatcher thread waits for requests
- For each request, choose an idle worker thread
- Worker thread uses blocking system calls to service web request



Pthread Management – Creating Threads

- The main() method comprises a single, default thread.
- **pthread_create**() creates a new thread and makes it executable.
- The maximum number of threads that may be created by a process is implementation dependent.
- Once created, threads are peers, and may create other threads.

Create Thread Call

```
#include <pthread.h>
int pthread_create(  pthread_t *thread,
                     pthread_attr_t *attr,
                     void *(*thread_function)(void *),
                     void *arg );
Returns 0 on success, or a positive error number on error.
```

1st arg – pointer to the identifier of the create thread

2nd arg – thread attributes. If null, then the thread is created with default attributes

3rd arg – pointer to the C function that the thread will execute once the thread is created

4th arg – the argument of the executed function returns 0 for success

Pthread Management – Terminating Threads

Several ways to terminate a thread:

- The thread is complete and returns
- The pthread_exit() method is called
- The pthread_cancel() method is invoked
- The exit() method is called

The pthread_exit() routine is called after a thread has completed its work and it no longer is required to exist.

Thread Termination

```
#include <pthread.h>

void pthread_exit (void *retval)
```

Retval specifies the return value for the thread.

The value pointed to by *retval* should not be located on the thread's stack, since the contents of that stack become undefined on thread termination.

If the main thread calls *pthread_exit()* or performing a **return** then the other threads continue to execute.

Thread IDs

```
#include <pthread.h>

pthread_t pthread_self (void)
```

Returns the thread ID of the calling thread.

A thread can obtain its own ID using *pthread_self()*

Thread Joining

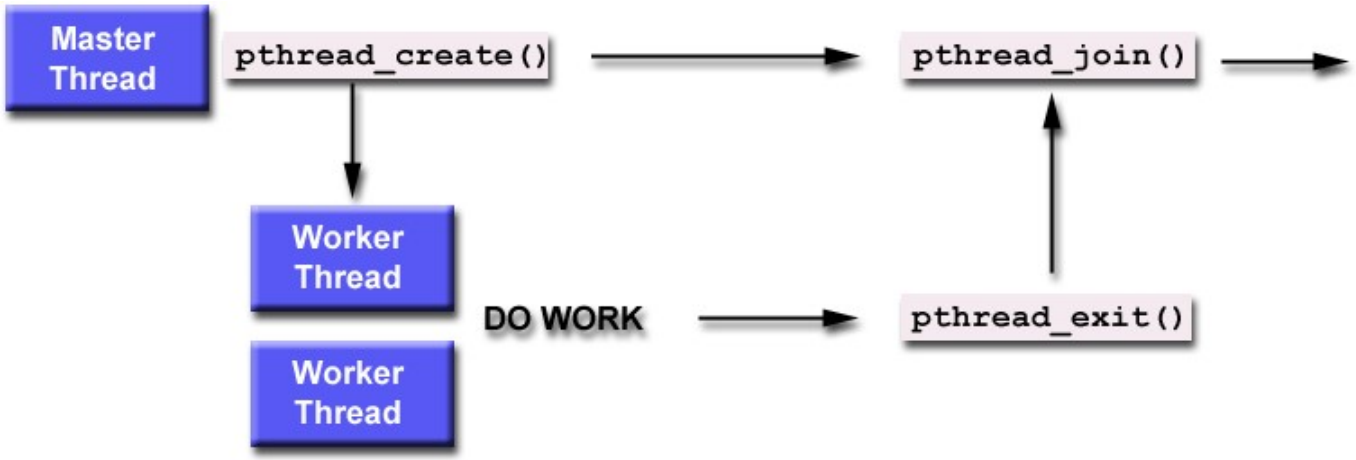
```
#include <pthread.h>

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

Returns 0 on success, or a positive number on error.

The *pthread_join()* function waits for the thread identified by *thread* to terminate.

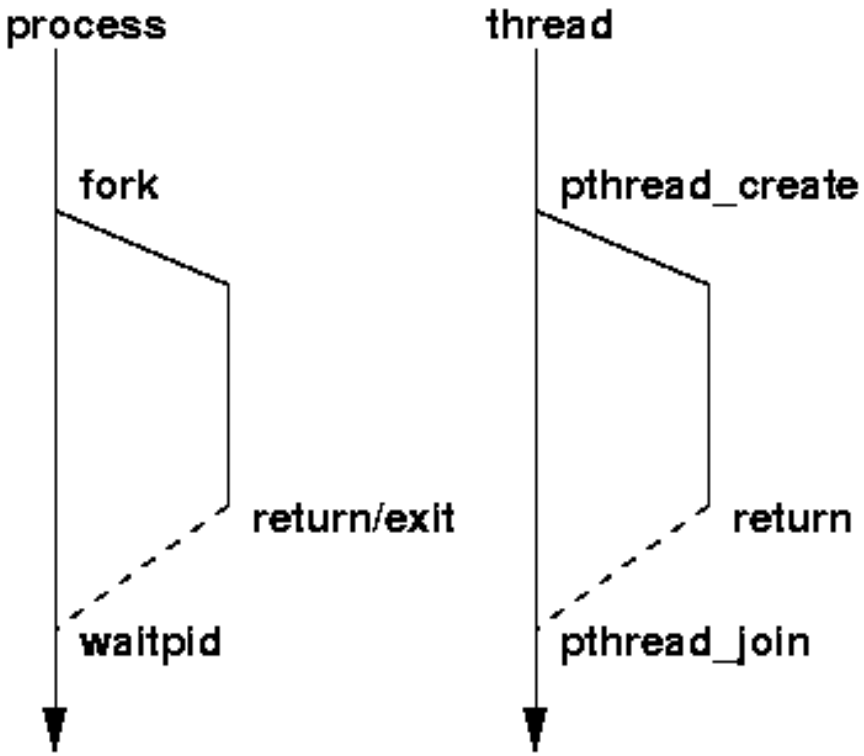
Thread Join (1 of 2)

- "Joining" is one way to accomplish synchronization between threads.



- The pthread_join() subroutine blocks the calling thread until the specified thread-id thread terminates.

Thread Join (2 of 2) – Compared with fork/waitpid



Advantages of Threads

The overhead for creating a thread is significantly less than that for creating a process

Multitasking, i.e., one process serves multiple clients

Switching between threads requires the OS to do much less work than switching between processes

Sharing data between threads is easier than processes

Drawbacks of Threads

Not as widely available as longer established features

Writing multithreaded programs require more careful thought

More difficult to debug than single threaded programs

For single processor machines (1 CPU), creating several threads in a program may not necessarily produce an increase in performance

Sample program (thread.c)

```
#include <stdio.h
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
int main(int argc, char **argv){
        pthread_t t1;
        int thread_id = 1;
       if ( (pthread_create(&t1, NULL, (void *)&worker (void *), &thread_id)) != 0) {
             printf("Error creating thread\n");
             exit(EXIT_FAILURE);
       }
       pthread_join(t1, NULL);
       return (EXIT_SUCCESS);
}
void *worker(void *a)  {
        int *cnt = (int *)a;
         printf("This is thread %d\n", *cnt);
         pthread_exit(0);
}
```

Special directions for compiling

- The use of **pthread.h** requires an additional flag at compile time, as it is not automatically included as are other include files.

- Example:

  gcc –o thread thread.c -lpthread

14-UNIX

POSIX Threads

The End