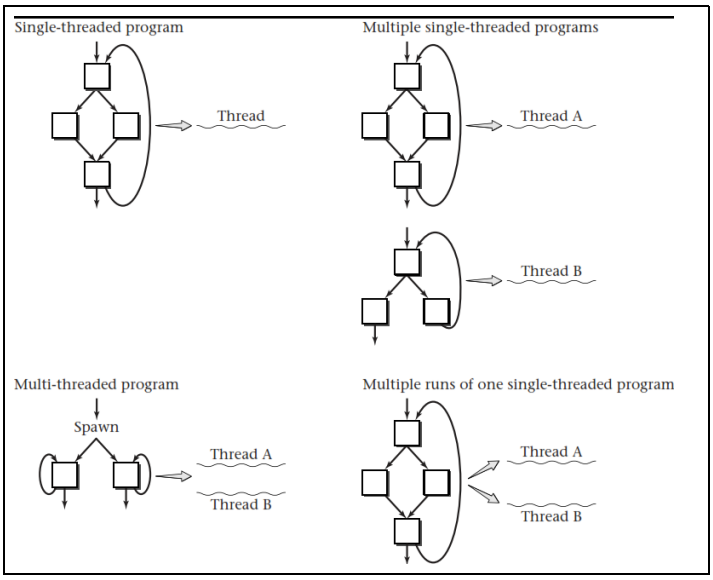
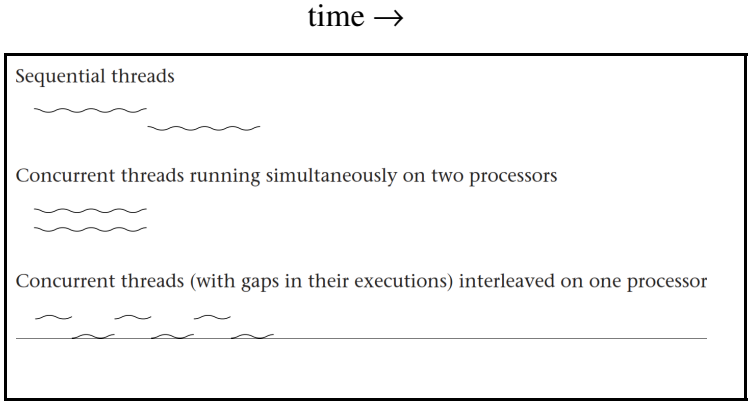
1) A *program* contains instructions. A *thread* is a sequence of computations from execution those instructions.



You have mostly written *single-threaded programs*. To get a multi-threaded program, your (C/C++, Java, etc.) program must explicitly *spawn* additional threads.



a) Why do you need multiple processors for concurrent threads to be running simultaneously?

b) Why might two concurrent threads interleaved on one processor be faster than sequential execution of the same two threads?

2) Figure 2.3 shows a simple multithreaded program in Java.

```
public class Simple2Threads {
 public static void main(String args[]){
   Thread childThread = new Thread(new Runnable(){
       public void run(){
         sleep(3000);
         System.out.println("Child is done sleeping 3 seconds.");
       }
     });
   childThread.start();
   sleep(5000);  // sleep 5000 millisecond or 5 seconds
   System.out.println("Parent is done sleeping 5 seconds.");
 }

 private static void sleep(int milliseconds){
   try{
     Thread.sleep(milliseconds);
   } catch(InterruptedException e){
     // ignore this exception; it won't happen anyhow
   }
 }
}
```

a) If the child thread sleeps 3 seconds and the parent thread sleeps 5 seconds, why does the whole program take only 5 seconds?

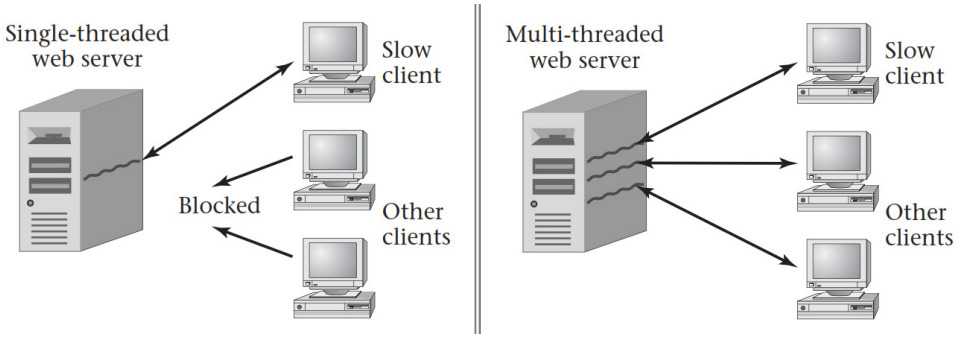
b) If you flipped the ".start() and sleep in the main, how long would the program run?

Figure 2.4 shows a simple multithreaded program in C.

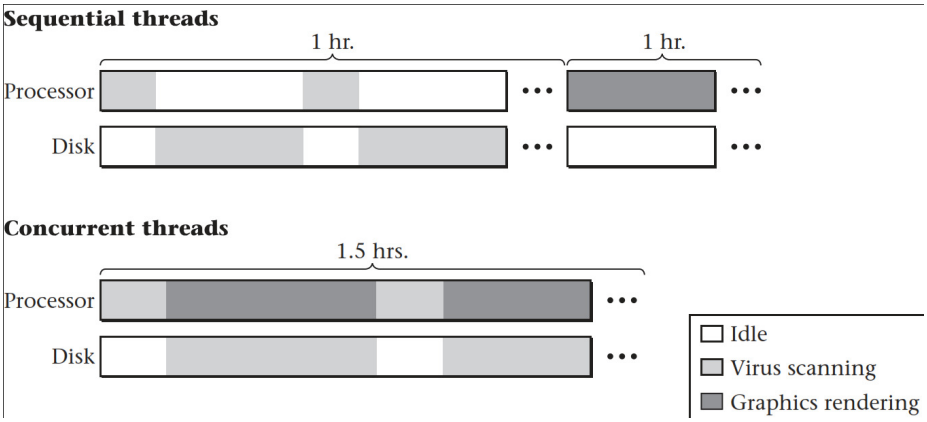
```
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>

static void *child(void *ignored){
  sleep(3);
  printf("Child is done sleeping 3 seconds.\n");
  return NULL;
}

int main(int argc, char *argv[]){
  pthread_t child_thread;
  int code;

  code = pthread_create(&child_thread, NULL, child, NULL);
  if(code){
    fprintf(stderr, "pthread_create failed with code %d\n", code);
  }
  sleep(5);
  printf("Parent is done sleeping 5 seconds.\n");
  return 0;
}
```

Some reasons for using concurrent threads:

- responsiveness - allows one thread to execute while another thread(s) are waiting for I/O or performing some long running computation
  Textbook example: web-server to supply many client computers with the web pages they requested.



- resource utilization - multiple concurrent threads can utilize multiple hardware resources. If one thread has no use for a piece of hardware, another my be able to utilize it. (e.g., utilizing processors in a muli-processor system, or one thread using I/O device while another is using the processor)



- tool for modularization - a complex system may naturally decomposed into a group of interacting threads, especially if you are modeling multiple independent objects.

3) A running program is called a *process* and the OS manages and restricts processes on the computer. As we saw, a process can call thread operations including thread creation, termination, synchronization primitives, scheduling, data management and process interaction.

All threads within a process share the same address space, so threads (pthread anyway) share:

All threads within a process share:	**Each thread has a unique:**
process instructions	thread ID
most data	set of registers, stack pointer
open files (descriptors)	stack for local variables, return addresses
signals and signal handlers	signal mask
current working directory	priority
User and group id	return value: errno (pthread functions return "0" if OK)

a) To switch the processor's focus from running one thread ("A") to running another thread ("B") from the same process (called a *thread switch*)?

b) What informatiom must be saved about thread "B" to resume it later?

c) Where should information about a thread be stored?

d) Outline the steps to switchFromTo(A, B)

e) When might a thread "voluntarily" *yield* usage of the processor?

f) How is a process protected from a thread that goes into an infinite loop?

g) How is a *process switch* different from a *thread switch*?