

4.6 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

Answer:

Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a “shell” program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

4.7 Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

Answer:

When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multithreaded solution would perform better even on a single-processor system.

4.8 Which of the following components of program state are shared across threads in a multithreaded process?

- a. Register values
- b. Heap memory
- c. Global variables
- d. Stack memory

Answer: The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

4.9 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?

Answer: A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate

processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system. involves saving and restoring the state of the registers.

4.10 In Chapter 3, we discussed Google's Chrome browser and its practice of opening each new website in a separate process. Would the same benefits have been achieved if instead Chrome had been designed to open each new website in a separate thread? Explain.

Answer:

No because the whole reason why a new process was created for each browser was because if a webpage crashed, it wouldn't crash the whole browser. If you use a thread for each webpage, then if a webpage crashes, it will bring down the whole application.

4.11 Is it possible to have concurrency but not parallelism? Explain.

Answer: Yes. Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

4.12 Using Amdahl's Law, calculate the speedup gain of an application that has a 60 per cent parallel component for (a) two processing cores and (b) four processing cores.

Answer:

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

S = serial portion (in this case 40%)

N = number of processing cores (in this case a) 2 b) 4)

Two processing cores = 1.43 speedup; four processing cores = 1.82 speedup

4.13 Determine if the following problems exhibit task or data parallelism:

a. Multi-threaded statistical program described in Exercise 4.21

ANS:

i. Task - multiple threads doing different things

b. Multi-threaded Sudoku validator describe in Project 1 in this chapter.

ANS:

- i. Task - multiple threads doing different things
- ii. Data - multiple threads doing the same thing on different data
- c. Multi-threaded sorting program describe in Project 2 in this chapter.

ANS:

- i. Data - each task does the same thing on different data
- d. Multi-threaded web server described in section 4.1.

ANS:

- i. Task - multiple task doing different things
- ii. Data - each request

4.14 A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between startup and termination, the program is entirely CPU bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

- How many threads will you create to perform the input and output? Explain.
- How many threads will you create for the CPU-intensive portion of the application? Explain.

Answer:

- It only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking. Creating additional threads provides no benefit. Thus, it makes sense to create a single thread for input and a single thread for output.
- Four. There should be as many threads as there are processing cores. Fewer would be a waste of processing resources, and any number > 4 would be unable to run

4.15 Consider the following code segment:

```
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
    fork();
    thread_create( . . . );
}
fork();
```

- a. How many unique processes are created?

b. How many unique threads are created?
created?(by the)

Answer:

There are six processes and two threads

4.16

As described in Section 4.7.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, other operating systems, such as Windows, treat processes and threads differently. Typically, such systems use a notation in which the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

Answer:

Yes - the Windows method SUCKS and is difficult to write on the OS level and debug on the user level. Don't use MS products.

(not my words by the way :))

More formally:

On one hand, in system where processes and threads are considered the same entities, some of the operating system code could be simplified. A scheduler for instance, can consider the different processes and threads on an equal footing without requiring special code to examine the threads associated with a process during every scheduling step.

On the other hand, this uniformity could make it harder to impose process-wide resource constraints in a direct manner. Instead, some extra complexity is required to identify which threads corresponds to which process and perform the relevant accounting tasks.

4.17 The program show in Figure 4.14 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

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

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

```
int value = 0;
```

```
void *runner(void *param); /* the thread */
```

```

int main(int argc, char *argv[])
{
    int pid;
    pthread_t tid;
    pthread_attr_t attr;
    pid = fork();
    if (pid == 0){ /* child process*/
        pthread_attr_init(&attr);
        pthread_create(&tid,&attr,runner,NULL);
        pthread_join(tid,NULL);
        printf("CHILD: value = %d" , value); /* LINE C */
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d" , value); /* LINE P */
    }
}

void *runner(void *param) {
    value = 5;
    pthread_exit(0);
}

```

Figure 4.14: C program for Exercise 4.4

ANS:

Output at LINE C is 5. Output at LINE P is 0.

4.18 Consider a multicore system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores in the system. Discuss the performance implications of the following scenarios.

- The number of kernel threads allocated to the program is less than the number of processing cores.
- The number of kernel threads allocated to the program is equal to the number of processing cores.
- The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.

Answer:

a) When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors.

b) When the number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernel thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle.

c) When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.

4.19 Pthreads provides an API for managing thread cancellation. The `pthread_setcancelstate()` function is used to set the cancellation state. Its prototype appears as follows:

```
pthread_setcancelstate(int state, int *oldstate)
```

The two possible values for the state are `PTHREAD_CANCEL_ENABLE` and `PTHREAD_CANCEL_DISABLE`.

Using the code segment shown in Figure 4.17, provide examples of two operations that would be suitable to perform between the calls to `disable` and `enable` thread cancellation.

Programming Problems 195

```
int oldstate;
pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldstate);
/* What operations would be performed here? */
pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, &oldstate);
```

Figure 4.17 C program for Exercise 4.19.

Answer:

Three examples:

- a. An update to a file
- b. A situation in which two write operations must both complete if either completes
- c. Essentially any operation that we want to run to completion

//extra one question

What are two differences between user-level threads and kernel-level threads?

Under what circumstances is one type better than other?

ANS:

(1) User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.

(2) On systems using either M: 1 or M: N mapping, user threads are scheduled by the thread

library and the kernel schedules kernel threads.

(3) Kernel threads need not be associated with a process whereas every user thread belongs to a

process. Kernel threads are generally more expensive to maintain than user threads as they must

be represented with a kernel data structure.