CS 537 - Introduction to Operating Systems

Project 5: Threads and Locking

Aman Rakesh Chadha
(9068354597)

1 Objectives

To get experience with the pthreads thread interface.

2 Part 1: Timing

You should write a simple program to measure the performance difference of creating a process as compared to creating a thread. This program, which you should name "threadperf", takes two parameters: the letter 't' or 'p' (for creating threads or processes) the number of threads or processes to create. Wait for each process/thread to terminate before starting the next one. The threads/processes should do no work, not even printing a message. They should return/exit immediately.

The program should execute a loop that (a) creates a thread with pthread_create() or a process with fork() and (b) waits for the thread/process to finish with pthread_join() or wait().

You should turn in this program with a readme file that includes measurements of the performance difference. This can be on any machine, including your private machines. The readme should state what kind of computer it is, how many processors it has, and how long the test takes for one million threads and one million processes. You can time your program with the Unix time command-line utility.

3 Part 2: Locking

You will be implementing a simple program to keep track of bank balances. Your program will simulate a random set of deposits and withdrawls.

Your program will read from the command line 5 integers:

- The number of threads to use
- The number of transactions per thread
- The maximum transaction size
- The initial bank balance
- A random seed

Example command line:
bank 10 1000 100 10000 52

uses 10 threads, 1000 transactions per thread, $100 maximum transaction, $10,000 initial balance, and a seed of 52.

Your program will then start separate threads. Each one will generate a random number between [- max transaction size, + max transaction size] with the C random_r()

function (which is safe to call from multiple threads), initialized with the srandom_r() function and the supplied random seed. You should call srandom_r() separately in each thread (this will make each thread generate the same sequence of transactions).

All values should be integers. If the number is negative, it is a withdrawal and your program must make sure the current balance is high enough. If the number is positive, it is a deposit and is always safe. Your program should acquire a lock, perform the transaction, then release the lock.

If the balance is too low for the withdrawal (i.e., the amount of the withdrawal is greater than the account balance), return an error but don't print an error message or set the balance to zero.

Once a thread has performed its transactions, it should exit. The program should terminate when all the thread have completed by printing out the final bank balance.

Your program should be named "bank". It should take the six parameters identified above, all as integers. The output of the program should be a single line:

Final bank balance: <balance>