**COSC 3346 Operating Systems  
  
Programming with pthreads**

**OBJECTIVE**

The purpose of this assignment is to become familiar with the pthreads API and write simple programs using pthreads.  The first program is already done for you.  You will have to modify it to obtain timing results and explore the use of data parallelism techniques for computing PI.  The second part of this programming assignment uses threads in a more functional parallelism paradigm for generating and displaying the Fibonacci sequence.  The third part of this programming project you will explore the use of thread synchronization without using semaphores.   For this programming project, you must use the departmental server riddler.tamucc.edu.

**PART A: Computing PI With Threads**

The first part of this assignment is to explore the use of threads in an application that computes PI.  See at the end of this document for the program you will be using for this part.  Create an appropriate Makefile to support the creation of the PI application (you can do this by modifying the Project1 Makefile).  Compile and run this program a few times to understand the basic operation and to verify that it does compute approximate PI.  Next, time your program and/or add appropriate timing system calls to your code.  Run the application varying the number of threads and number of sample points.  You may get a core dump when you exceed the number of threads permitted (note that value).  Record the time(s) it takes your program to execute when changing the number of threads and number of sample points.

Create a plot of time vs. number of threads when the number of sample points is fixed (Use y-axis for time and x-axis for number of threads).  Notice, the larger is the number of sample points the better is the accuracy of PI calculation. Use an Excel like application to create your plot.

The name of this program must be proj2a.c or proj2a.cpp.

**PART B: Computing Fibonacci Numbers With Threads**

This programming project is based on a problem given in our textbook.  The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8,....  Formally, it can be expressed as:

fib0 = 0  
fib1 = 1  
fibn = fibn-1 + fibn-2

Write a multithreaded program that generates the Fibonacci series using the pthreads library.  This program should work as follows:  The user will enter on the command line the number of Fibonacci numbers that the program will generate.  The program will then create a separate thread that will generate the Fibonacci numbers placing the sequence in data that is shared by the threads (an array or vector is probably the most convenient data structure).  When the thread finishes execution, the parent thread will output the sequence generated by the child thread.  Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish, using the techniques discussed in the class. The name of this program must be proj2b.c or proj2b.cpp.

**PART C: Printing Words With sched\_yield()**

Write a program that accepts a phrase of unspecified length on the command line. For example:

prompt% vowcon Operating Systems Class at TAMUCC

The main() in this program has to create two threads running functions (vow and cons). The threads should take turns printing the respective words of the phrase supplied on the command line. The vow thread should print all the words that start with a vowel and the cons thread should print all words starting with a consonant. Note that the main thread (running main()) should not print anything itself, the output should be supplied by the threads that it creates. The order of words in the phrase should not be changed in the printout. Your program should work for any phrase of any reasonable length, not just the one given in the example. The output of your program should look similar to the following

$ ./proj2c Operating Systems Class at TAMUCC

vow: Operating

cons: Systems

cons: Class

vow: at

cons: TAMUCC

In this part you are \*not allowed\* to use synchronization primitives such as mutexes for thread coordination. You will have to investigate some of the other pthread functions available to you for this project. Look for examples online.

The name of this program must be proj2c.c or proj2c.cpp.

**OTHER USEFUL INFORMATION**

There are a few systems calls to get and record timing values in user programs available in C/C++.  The 'time' system command can also be used as a system call.  Other timing operations such as ctime(), gettimeofdday(), ftime(), and time() are system functions and can only be used in a program.  Carefully read the man pages for these commands and system calls. You may check online how to use these functions to measure CPU time, execution time, elapsed time, and so on.

A simple command to compile a program (prog.c) with pthread library:

prompt$> gcc prog.c -lpthread

It will create an executable a.out.

**ASSESSMENT AND GRADING**

This is an individual assignment.  Write a well-structured, well-documented computer program using C/C++ and the supporting files provided to you.  Use procedures and functions.  You may use the STL in your solution, if needed.  This project is worth 100 points.

**PROJECT SUBMISSION**

1. Your work should be submitted as a zip file on Blackboard. This file should restore the files in a directory, named after your username, an underscore, and the project number (Ex: dkar\_proj2zip) containing the appropriate files and/or answers to the question(s), if any.

2. Unless otherwise noted in the assignment notice, your answer must compile/run/work on riddler.tamucc.edu.

|  |
| --- |
| Code for Part A: compute\_pi\_montecarlo.c |
| #include <pthread.h>  #include <stdlib.h>  #include <stdio.h>  #define MAX\_THREADS 512  void \*compute\_pi( void \* );  int sample\_points;  int total\_hits;  int total\_misses;  int hits[ MAX\_THREADS ];  int sample\_points;  int sample\_points\_per\_thread;  int num\_threads;  int main( int argc, char \*argv[] )  {  /\* local variables \*/  int ii;  int retval;  pthread\_t p\_threads[MAX\_THREADS];  pthread\_attr\_t attr;  double computed\_pi;  /\* initialize local variables \*/  retval = 0;  pthread\_attr\_init( &attr );  pthread\_attr\_setscope( &attr, PTHREAD\_SCOPE\_SYSTEM );  /\* parse command line arguments into sample points and number of threads \*/  /\* there is no error checking here!!!!! \*/  sample\_points = atoi(argv[1]);  num\_threads = atoi(argv[2]);  /\* uncomment this block if you want interactive input!!!! \*/  /\* if so...comment out the two statements above \*/  /\*  printf( "Enter number of sample points: " );  scanf( "%d", &sample\_points );  printf( "Enter number of threads: " );  scanf( "%d%", &num\_threads );  \*/  total\_hits = 0;  sample\_points\_per\_thread = sample\_points / num\_threads;  for( ii=0; ii<num\_threads; ii++ )  {  hits[ii] = ii;  pthread\_create( &p\_threads[ ii ], &attr, compute\_pi, (void \*) &hits[ii] );  }  for( ii=0; ii<num\_threads; ii++ )  {  pthread\_join( p\_threads[ ii ], NULL );  total\_hits += hits[ ii ];  }  computed\_pi = 4.0 \* (double) total\_hits / ((double) (sample\_points));  printf( "Computed PI = %lf\n", computed\_pi );  /\* return to calling environment \*/  return( retval );  }  void \*compute\_pi( void \*s )  {  int seed;  int ii;  int \*hit\_pointer;  int local\_hits;  double rand\_no\_x;  double rand\_no\_y;  hit\_pointer = (int \*) s;  seed = \*hit\_pointer;  local\_hits = 0;  for( ii=0; ii < sample\_points\_per\_thread; ii++ )  {  rand\_no\_x = (double) (rand\_r( &seed ))/(double)RAND\_MAX;  rand\_no\_y = (double) (rand\_r( &seed ))/(double)RAND\_MAX;  if(((rand\_no\_x - 0.5) \* (rand\_no\_x - 0.5) +  (rand\_no\_y - 0.5) \* (rand\_no\_y - 0.5)) < 0.25)  local\_hits++;  seed \*= ii;  }  \*hit\_pointer = local\_hits;  pthread\_exit(0);  } |