The University of Texas at Arlington

CSE 3320 - Spring 2020
Operating Systems
Project 2 - POSIX Thread Programming

Instructor: Jia Rao

Points Possible: 100
Handed out: Feb. 28, 2020
Due date: 11:59 pm, Thursday, Mar. 19, 2020

Introduction

The purpose of this project is to practice `Pthread` programming by solving various problems. The objectives of this project is to learn:

1. Get familiar with the `Pthread` creation and termination.
2. How to use mutexes and conditional variables in `Pthread`.
3. How to design efficient solutions for mutual exclusion problems.

Project submission

For each project, create a gzipped file containing the following items, and submit it through Blackboard.

1. A report that briefly describes how did you solve the problems and what you learned.
2. The POSIX thread programming codes and files containing your test cases.

Assignments

Assignment 1 (40 pts)

Given two character strings `s1` and `s2`. Write a `Pthread` program to find out the number of substrings, in string `s1`, that is exactly the same as `s2`. For example, suppose `number_substring(s1, s2)` implements the function, then `number_substring(“abcdab”, “ab”)` = 2, `number_substring(“aaa”, “a”)` = 3, `number_substring(“abac”, “bc”)` = 0. The size of `s1` and `s2` ($n1$ and $n2$) as well as their data are input by users. Assume that $n1$ mod $NUM\_THREADS = 0$ and $n2 < n1/NUM\_THREADS$.

The following is a sequential solution of the problem. `read_f()` reads the two strings from a file named "`string.txt` and `num_substring()` calculates the number of substrings.

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

#define MAX 1024

int total = 0;
int n1,n2;
char *s1,*s2;
FILE *fp;

int readf(FILE *fp)
{
  if((fp=fopen("strings.txt", "r"))==NULL){
    printf("ERROR: can't open string.txt!\n");
    return 0;
  }
  s1=(char *)malloc(sizeof(char)*MAX);
  if(s1==NULL){
    printf("ERROR: Out of memory!\n");
    return -1;
  }
  s2=(char *)malloc(sizeof(char)*MAX);
  if(s1==NULL){
    printf("ERROR: Out of memory\n");
    return -1;
  }
  /*read s1 s2 from the file*/
  s1=fgets(s1, MAX, fp);
  s2=fgets(s2, MAX, fp);
  n1=strlen(s1);  /*length of s1*/
  n2=strlen(s2)-1; /*length of s2*/
  if(s1==NULL || s2==NULL || n1<n2)  /*when error exit*/
    return -1;
}

int num_substring(void)
{
  int i,j,k;
  int count;

  for (i = 0; i <= (n1-n2); i++){
    count=0;
    for(j = i,k = 0; k < n2; j++,k++){  /*search for the next string of size of n2*/
      if (*(s1+j)!=*(s2+k)){
        break;
      }
      else
        count++;
      if(count==n2)
        total++;    /*find a substring in this step*/
    }
  }
  return total;
}

int main(int argc, char *argv[])
{
  int count;

  readf(fp);
  count = num_substring();
```

```
  printf("The number of substrings is: %d\n", count);
  return 1;
}
```

Write a parallel program using `Pthread` based on this sequential solution.

HINT: Strings s1 and s2 are stored in a file named "string.txt". String s1 is evenly partitioned for NUM_THREADS threads to concurrently search for matching with string s2. After a thread finishes its work and obtains the number of local matchings, this local number is added into a global variable showing the total number of matched substrings in string s1. Finally this total number is printed out. You can find an example of the "string.txt" in the attached source code.

Assignment 2 (35 pts)

Use *condition variables* to implement the producer-consumer algorithm. Assume two threads: one producer and one consumer. The producer reads characters one by one from a string stored in a file named "message.txt", then writes sequentially these characters into a circular queue. Meanwhile, the consumer reads sequentially from the queue and prints them in the same order. Assume a buffer (queue) size of 5 characters. Write a `Pthread` program using condition variables.

Assignment 3 (25 pts)

Write two micro-benchmarks to quantify the costs of context switch between multiple processes and multiple threads. Learn from the `lat_ctx` benchmark from the lmbench benchmark suite about how to measure the context switch cost between multiple processes. Understand how this benchmark works and adapt it to write your own benchmarks. Configure the number of vCPUs to more than one but run all processes/threads on one vCPU. Vary the number of processes/threads to study if the context switch cost varies. Below is a related work on quantifying the cost of context switch.
`https://www.cs.rochester.edu/~kshen/papers/expcs2007.pdf`

Tips
(1) Shutdown your VM and change to VM setting to use 4 vCPUs.

(2) Verify that your VM has 4 vCPUs:

```
$ cat /proc/cpuinfo
```

You should have 4 CPUs (processor: 0-3).