CS3113fa17 Homework 4

Due 11/16/2017 11:45pm CST (50 Points)

Desciption

The goal of this project is to observe and understand the performance of Semaphores across kernel execution elements. You will also get a better understanding of the pthread libraries and build for intuition of the performance of threads.

Take the `thread_incr_psem.c` code and modify it to take `N` threads in addition to the existing `num-loops` parameter. This code is located in your instance and in the Linux Programming Interface Textbook code (/projects/tlpi-dist/psem/). You may aslo located it with the `find` command. By default, the code only executes two threads. Change the code to make this program use an arbitraty amount of threads to perform the counts. You should be able to run it with the following usage: `./thread_incr_psem_threads 1000 4`.

We will use the `time` to capture the performance times for several arrangements of threads and loops. Install the updated version with `apt install time`. For more information about the time command, see this website: https://www.lifewire.com/command-return-time-command-4054237. You also need to install the following programs on your instance:

```bash
apt-get install -y time tmux sqlite3 libsqlite3-dev
apt-get install -y libsqlite3-0 libsqlite3-0-dbg
apt-get install -y python3 ipython3 python3-pip
apt-get install -y python3-tk ssh
pip3 install --upgrade pip
pip3 install pip pandas numpy matplotlib
```

Use the script below to gather the data for your experiment. The script below will run your code for 2, 4, 8, and 16 threads and also for loops of size 20000000, 40000000, 80000000, and 160000000. The program takes the executable name and the a csv file for output. For example, if you save this script as runtest.bashsh, you can run this code using the command below.
`runtest.bash ./t_psem experiment`date +%Y%m%d%M`.csv` . Be sure that your `runtest.bash` file is executable. The output of the code is the name of your csv file and an sqlite database file (`*.db`), a pdf and png of the run, and the csv of the exerperiments with runs averaged and the standard deviation computed. There is an additional csv file that is prefixed with raw and the name of your program; this file contains a row for each run of your program, it has the following format:

```
total cores,
loop size,
threads count,
real_time average,
real_time std deviation,
user_time average,
user_time std deviation,
kernel_time average,
kernel_time std deviation
```

Note that the script below only plots the *real time.* You should still understand the difference between *real time*, *user time*, and *kernel time*. This difference is discussed in you text book. You may also look for other resources for additional explanations. Below is the script you can use to run the code and produce graphs your graphs. Installing the packages above will allow you to execute the script below. The script uses some python scripting, database scripts, and some commandline Fu to help generate the graph. Examining the code may be helpful for you, but you only need to add it to an exexutable file and run it.

```bash
#!/bin/bash
## Usage: runtest.bash ./code experiment`date +%Y%m%d%M`.csv

EXE=$1
CSVFILE=$2
CORES=$(grep -c '^processor' /proc/cpuinfo)
```

```
for threads in 2 4 8 16
do
  for loops in 20000000 40000000 80000000 160000000
  do
    /usr/bin/time -f "$CORES, $loops, $threads, %e, %U, %S" \
        --append --quiet --output=raw_$2 \
        $1 $loops $threads
    /usr/bin/time -f "$CORES, $loops, $threads, %e, %U, %S" \
        --append --quiet --output=raw_$2 \
        $1 $loops $threads
    /usr/bin/time -f "$CORES, $loops, $threads, %e, %U, %S" \
        --append --quiet --output=raw_$2 \
        $1 $loops $threads
  done
done

# Need to calculate stddev, Installing sqlite extension
rm extension-functions.c*
wget http://sqlite.org/contrib/download/extension-functions.c/\
download/extension-functions.c?get=25
mv extension-functions.c?get=25 extension-functions.c
rm libsqlitefunctions.so
# https://stackoverflow.com/a/16682644/235820
gcc -fPIC -shared extension-functions.c -o libsqlitefunctions.so -lm
mv libsqlitefunctions.so /usr/local/lib/


sqlite3 -batch $2.db <<EOF
-- create the table
.load /usr/local/lib/libsqlitefunctions.so
CREATE TABLE experiment (
    cores integer,
    loops integer,
    threads integer,
    real_time real,
    user_time real,
    kernel_time real
);
.separator ","
-- load the data file to the table
.import raw_${2} experiment
-- create a new table with the errors
CREATE TABLE experiment_error AS
    SELECT cores, loops, threads,
        avg(real_time) as avg_rt, stdev(real_time) as std_rt,
        avg(user_time) as avg_ut, stdev(user_time) as std_ut,
        avg(kernel_time) as avg_kt, stdev(kernel_time) std_kt
    FROM experiment
    GROUP BY cores, loops, threads;
EOF

# Write data to new file
sqlite3 -csv $2.db "SELECT * from experiment_error;" > $2

# Use python to produce plot of results for $CORES
## x -- loops
## y -- time (seconds)
## color -- thread
## dotted --

python3 - <<EOF
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import os
import pandas as pd
import sys

cores = '${CORES}'
csv_file = '${2}'

# https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html
df = pd.read_csv(csv_file, names=['cores',
                 'loops',
                 'threads',
                 'avg_rt',
                 'std_rt',
                 'avg_ut',
                 'std_ut',
                 'avg_kt',
                 'std_kt'])

# split by threads
df2 = df.query('threads == 2')
df4 = df.query('threads == 4')
df8 = df.query('threads == 8')
df16 = df.query('threads == 16')
```

```python
# https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html
title = "Semaphore addition time for multiple loops and threads on ${CORES}"
df2.plot(x='loops', y='avg_rt', yerr='std_rt',
        title=title, color='red' )
df4.plot(x='loops', y='avg_rt', yerr='std_rt',
        title=title, color='green' )
df8.plot(x='loops', y='avg_rt', yerr='std_rt',
        title=title, color='blue')
df16.plot(x='loops', y='avg_rt', yerr='std_rt',
        title=title, color='yellow' )

# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.fill_between.html
plt.fill_between(df2['loops'],
                 df2['avg_rt'] + df2['std_rt'],
                 df2['avg_rt'] - df2['std_rt'],
                 interpolate=False,
                 color='red',
                 alpha=0.2)
plt.fill_between(df4['loops'],
                 df4['avg_rt'] + df4['std_rt'],
                 df4['avg_rt'] - df4['std_rt'],
                 interpolate=False,
                 color='green',
                 alpha=0.2)
plt.fill_between(df8['loops'],
                 df8['avg_rt'] + df8['std_rt'],
                 df8['avg_rt'] - df8['std_rt'],
                 interpolate=False,
                 color='blue',
                 alpha=0.2)
plt.fill_between(df16['loops'],
                 df16['avg_rt'] + df16['std_rt'],
                 df16['avg_rt'] - df16['std_rt'],
                 interpolate=False,
                 color='yellow',
                 alpha=0.2)

plt.ylabel('Time (s)')
plt.xlabel('Loops')

plt.legend(['Threads 2', 'Threads 4', 'Threads 8', 'Threads 16'],
           loc='upper left')

# Save the two figures
plt.savefig("{}_rt.png".format(csv_file), bbox_inches='tight')
plt.savefig("{}_rt.pdf".format(csv_file), bbox_inches='tight')
EOF
```

The code you should modify is available on the book website (thread_incr_psem). It is also in your instance in the `/projects/tlpi-dist/psem/` directory. You may decide on how to compile and execute the file. The script above will simply execute that code, compute timings, and create graphs.

To view and download the graphs that are created in your instance, you can create a small Python file server with one command line. This way you can point to the webbrowser and download the processed files. Becareful, this is a security risk.

```bash
# Creates a simple file server.
# View by pointing browser to https://<external ip>:8889
python3 -m http.server 8889
```

Run the code and observe the plotted results. Peruse the trends that appear in the results and write write a 1-2 page paper describing your code, and the trends that you observed. Include anything that surprised you.

Bonus

Edit the number of processors in your instance or create three additional *vm instances* with cores of size 4, 8, and 16. Run the same scripts on all four machine configurations. Plot and report the results. Becareful not to leave these large instances running for a long period of time.

Grading

The paper you submit shoud include the following:

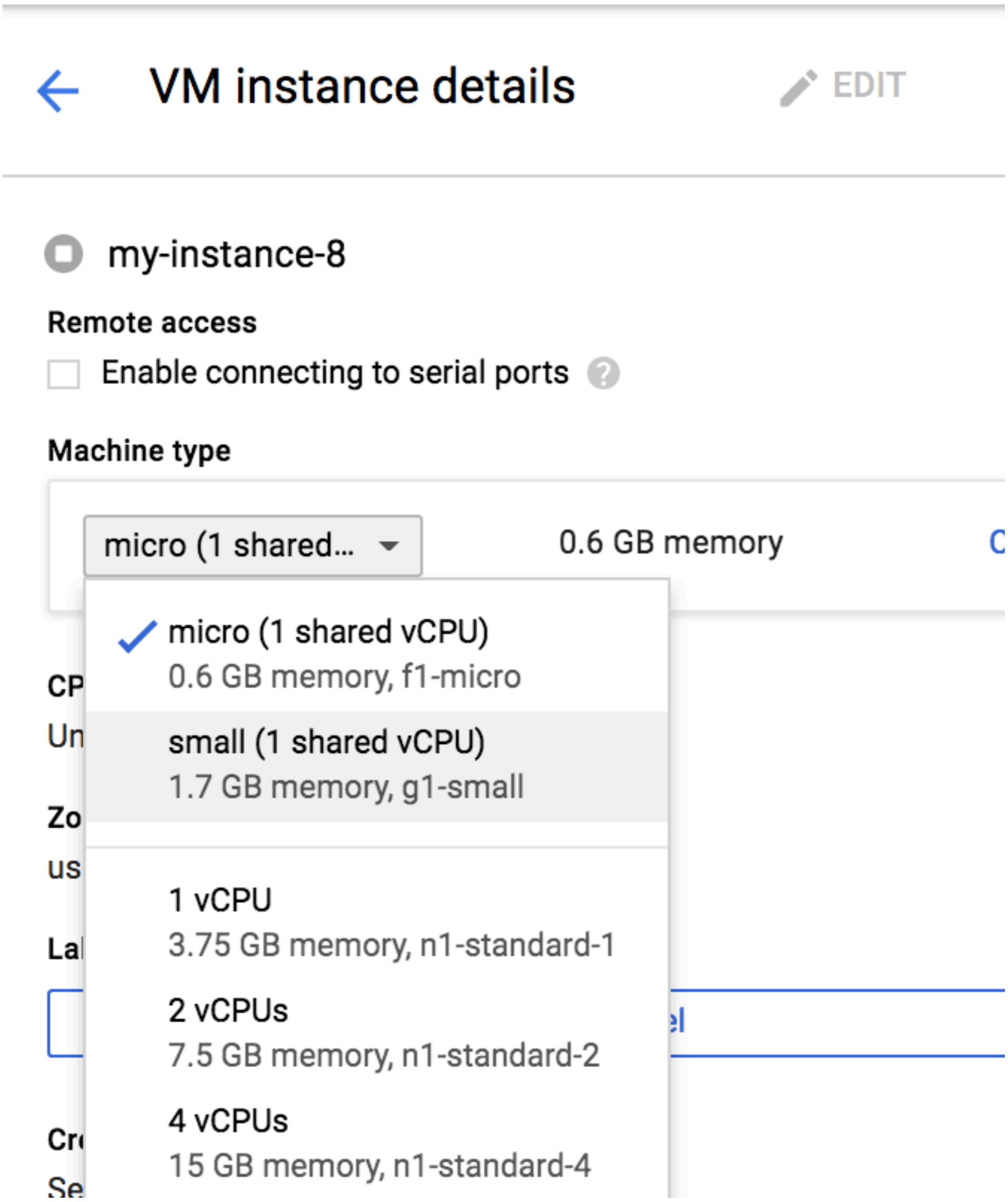
- A description of how you wrote and compiled your code.
- A set of all the assumption you to run the code.
- Any bugs or corner cases you have handled.
- Definition of semaphores
- Definitions of "real time", "kernel time", "user time".
- Decriptions of trends of the plots.
- Explaination ofthe trend results.

The point breakdown is below. Any errors and insuffiencies in any part of the criteria will result in a loss of points. Both the write-up and code should be submitted. Please submit only the text files (.c, .h, .txt) or .pdf files of your write up and code. There is no need to include makefile or build files.

Task	Percent
Code created correctly	25%
Write-up includes full discussion	75%
Bonus discussion in included	20% (10 pts)
Total	**120%**

Reconfiguring your instance

Note: You can edit the machine configuration on the vm configuration screen.



VM instance details

EDIT

my-instance-8

Remote access

☐ Enable connecting to serial ports

Machine type

micro (1 shared… ▾   0.6 GB memory

✓ micro (1 shared vCPU)
0.6 GB memory, f1-micro

small (1 shared vCPU)
1.7 GB memory, g1-small

1 vCPU
3.75 GB memory, n1-standard-1

2 vCPUs
7.5 GB memory, n1-standard-2

4 vCPUs
15 GB memory, n1-standard-4