Operating Systems Assignment 3 (CLO 1, PLO1, Total Marks 25)

Question 1 (20)

In this homework, you'll measure the **costs of a system call and context switch**. Measuring the cost of a system call is relatively easy. For example, you could repeatedly call a simple system call (e.g., performing a 0 byteread), and time how long it takes; dividing the time by the number of iterations gives you an estimate of the cost of a system call.

One thing you'll have to take into account is the precision and accuracy of your timer. A typical timer that you can use is gettimeofday(); read the man page for details. What you'll see there is that gettimeofday(). returns the time in microseconds since 1970; however, this does not mean that the timer is precise to the microsecond. Measure back-to-back calls to gettimeofday() to learn something about how precise the timer really is; this will tell you how many iterations of your null system-call test you'll have to run in order to get a good measurement result. gettimeofday() is not precise enough for you, you might look into using the rdtsc instruction available on x86 machines.
Measuring the cost of a context switch is a little trickier. The lmbench benchmark does so by running two processes on a single CPU, and setting up two UNIX pipes between them; a pipe is just one of many ways processes in a UNIX system can communicate with one another. The first process then issues a write to the first pipe, and waits for a read on the second; upon seeing the first process waiting for something to read from the second pipe, the OS puts the first process in the blocked state, and switches to the other process, which reads from the first pipe and then writes to the second. When the second process tries to read from the first pipe again, it blocks, and thus the back-and-forth cycle of communication continues. By measuring the cost of communicating like this repeatedly, lmbench can make a good estimate of the cost of a context switch. You can try to re-create something similar here, using pipes, or perhaps some other communication mechanism such as UNIX sockets.

You can take guidance from lmbench code at the following link:
https://github.com/intel/lmbench/blob/master/src/lat_ctx.c

One difficulty in measuring context-switch cost arises in systems with more than one CPU; what you need to do on such a system is ensure that your context-switching processes are located on the same processor. Fortunately, most operating systems have calls to bind a process to a particular processor; on Linux, for example, the sched_setaffinity() call is what you're looking for. By ensuring both processes are on the same processor, you are making sure to measure the cost of the OS stopping one process and restoring another on the same CPU.

(Solution)

Cost of syscalls:

Run **sysCallCost.c**

Terminal Output:

```
Start time is '899977'
End   time is '986993' in 100000 number of syscalls
Time it took is '0.870160'
[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngin
e-In-3pquhccv.lgq" 1>"/tmp/Microsoft-MIEngine-Out-o2hco2bl.5hc"
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/008_HA3$
```

-> 0.87016 us on average for each.

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

double findSystemCallCost()
{
//For storing the file descriptors of the pipe
int pipefds[2];

//Creating a buffer; not sure if function would still perform
//functionality if given null instead of this buffer
char buffer[1];

//Creating a pipe, returning file descriptors, and error check
if(pipe(pipefds) ==-1) { perror("pipe"); exit(EXIT_FAILURE);}
close(pipefds[1]);

//Instantiation of an object of timeval structure defined in sys/time.h
struct timeval current_time;

//Profiling start time
gettimeofday(&current_time, NULL);

//Variables for tracking start and end time
double start_time = current_time.tv_usec;
double end_time = 0;

//Running a loop of system call
int loop = 100000;
for (int i=0; i<loop; i++)
{int j = read(pipefds[0], buffer, 0);}

//Getting end time
gettimeofday(&current_time, NULL);
end_time = current_time.tv_usec;

//Closing read of pipe
close(pipefds[0]);
//Printing log
printf("Start time is '%d'\n", (int)start_time);
printf("End time is '%d' in %d number of syscalls \n", (int)end_time, loop);
printf("Time it took is '%f'\n", ((end_time-start_time)/(loop)));
};

int main(){
findSystemCallCost();
return EXIT_SUCCESS;}
```

Cost of Context Switch:

Run **contSwitchCost.c**

Terminal Output:

```
ChildTime[0] -> 458550
ChildTime[1] -> 458718
ChildTime[2] -> 458903
ChildTime[3] -> 459057
ChildTime[4] -> 459227
ChildTime[5] -> 459395
ChildTime[6] -> 459515
ChildTime[7] -> 459692
ChildTime[8] -> 460169
ChildTime[9] -> 460699
ChildTime[10] -> 461252
ChildTime[11] -> 461811
ChildTime[12] -> 462397
ChildTime[13] -> 462974



ParentTime[0] -> 458556
ParentTime[1] -> 458756
ParentTime[2] -> 458949
ParentTime[3] -> 459095
ParentTime[4] -> 459275
ParentTime[5] -> 459438
ParentTime[6] -> 459546
ParentTime[7] -> 459747
ParentTime[8] -> 460223
ParentTime[9] -> 460745
ParentTime[10] -> 461292
ParentTime[11] -> 461851
ParentTime[12] -> 462430
ParentTime[13] -> 463012
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngin
e-In-jymy3d24.iq3" 1>"/tmp/Microsoft-MIEngine-Out-3nnueo1x.xz2"
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/008_HA3$
```

Run **help.c** in helping directory. Which parses from **log.txt** and highlights important characteristics of observation.

Terminal Output:

```
Diff is 6
Diff is 38
Diff is 46
Diff is 38
Diff is 48
Diff is 43
Diff is 31
Diff is 55
Diff is 54
Diff is 46
Diff is 40
Diff is 40
Diff is 33
Diff is 38
Total Difference: 556
Highest Difference: 55
Lowest Difference: 6
Average Difference: 39.71
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngin
e-In-v3dphdyg.rah" 1>"/tmp/Microsoft-MIEngine-Out-frn4irn2.rcv"
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/008_HA3$
```

-> On average it is taking 39.71 us for each context switch.

Source Code:

```
#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <time.h>

int main()
{
int pipefds[2];
char*pin;
char buffer[10];
//Instantiation of an object of timeval structure defined in sys/time.h
struct timeval time;
cpu_set_t set;

if(pipe(pipefds) ==-1) { perror("pipe"); exit(EXIT_FAILURE);}
CPU_ZERO(&set);
pid_t pid = fork();
if (pid == 0)
{
CPU_SET(0, &set);
// in child process
sleep(1);
// PIN to send
pin = "This is a test\0";
int childTime[14];
// close read fd
close(pipefds[0]);
for(int i = 0; i<14; i++)
{
int j = write(pipefds[1], pin+i,1); // write PIN to pipe
gettimeofday(&time, NULL);
childTime[i]=time.tv_usec;
sleep(1);
//printf("Child time is %d\n", time.tv_usec);
//printf("Child sent PIN '%c', chars write %d\n", pin[i], j);
}
for(int i = 0; i<14; i++)
{
printf("ChildTime[%d] -> %d \n", i, childTime[i]);
}
printf("\n \n \n");
sleep(5);
```

```
close(pipefds[1]);
exit(EXIT_SUCCESS);
}

else
{
CPU_SET(0, &set);
// in main process
close(pipefds[1]); // close write fd
int parentTime[14];

for (int i=0;i<14;i++)
{
int j = read(pipefds[0], buffer,1); // read PIN from pipe
gettimeofday(&time, NULL);
parentTime[i]=time.tv_usec;
//printf("Parent time is %d\n", time.tv_usec);
//printf("Parent received PIN '%s', chars read %d\n", buffer, j);
}

close(pipefds[0]); // close read fd
wait(NULL); // wait for child process to finish

for(int i = 0; i<14; i++)
{
printf("ParentTime[%d] -> %d \n", i, parentTime[i]);
}
}
return EXIT_SUCCESS;
}
```