

MEASURING THE TIME TAKEN BY SYSTEM CALLS AND PROCEDURAL CALLS

LAB 7

SCENARIO 1

To compare the time taken by system calls in comparison with function calls. For better understanding, I have written 2 different functions:

1. **Foo()** – a simple function which only has a return statement
2. **findFactorial(int n)** – a function that returns the factorial of an input n

CODE

```
#include <sys/time.h>

#include <unistd.h>

#include <assert.h>

#include <stdio.h>

int foo()
{
    return(10);
}

int findFactorial(int n)
{
    int i;
    int factorial = 1;
    for(i = n; i >= 1; i--)
    {
        factorial *= i;
    }
    return factorial;
}

long nanosec(struct timeval t) /* Calculate nanoseconds in a timeval structure */
{
```

```

    return((t.tv_sec*1000000+t.tv_usec)*1000);
}

int main()
{
    int i,j,res;
    long N_iterations=1000000; /* A million iterations */
    float avgTimeSysCall, avgTimeFuncCall, avgTimeFactFuncCall[3];
    struct timeval t1, t2;

    res=gettimeofday(&t1,NULL);
    assert(res==0);
    for (i=0; i<N_iterations; i++) {
        j=getpid();
    }
    res=gettimeofday(&t2,NULL);
    assert(res==0);
    avgTimeSysCall = (nanosec(t2) - nanosec(t1))/(N_iterations*1.0);
    res=gettimeofday(&t1,NULL);
    assert(res==0);
    for (i=0; i<N_iterations; i++) {
        j=foo();
    }
    res=gettimeofday(&t2,NULL);
    assert(res==0);
    avgTimeFuncCall = (nanosec(t2) - nanosec(t1))/(N_iterations*1.0);
    printf("\n\nAverage time for System call getpid : %f\n",avgTimeSysCall);
    printf("\n\nAverage time for Function call : %f\n\n",avgTimeFuncCall);

    res=gettimeofday(&t1,NULL);
    int arr[] = {100, 50, 10};
    for(i = 0; i < 3; i++)

```

```
vmdhruv@ubuntu:~/Documents$ gcc -o sysVsProc sysVsProc.c
vmdhruv@ubuntu:~/Documents$ ./sysVsProc

Average time for System call getpid : 10.620000

Average time for Function call : 7.200000

Invoking function to find 100 factorial
Invoking function to find 50 factorial
Invoking function to find 10 factorial

Average time for Factorial function call [100,50,10]:

                [0.018000,          0.047000,          0.077000]
```

Thus we can see that **system calls involve context switching from user mode to kernel mode and back**. Hence they require the most amount of time. **Procedure calls** on the other hand **operate only in the user mode** and do not have context switches. Thus, they are faster.

In the case of the **findFactorial() function**, we see that factorial(100) takes lesser time than factorial(50) and factorial(10). We expected the reverse order, since factorial(100) has the most number of recursive calls among the three. However, by the **paging algorithms**, the part of the code referenced again and again is stored in the fastest storage, when compared to the parts of the code not called very often. Hence the output shows that factorial(100) is computed faster than factorial(50), which is in turn faster than factorial(10).

SCENARIO 2

Instead of using the `gettimeofday()` function as used in the previous question, we use the count clock cycles of the CPU to find the time taken by the process. This can be done by invoking the function **`clock()`** from the **<time.h> header file**.

CODE 2

```
/* Program to demonstrate time taken by function fun() */
#include <stdio.h>
#include <time.h>

// A function that terminates when enter key is pressed
void fun()
{
    printf("\nfun() starts \n");
    printf("Press enter to stop fun \n");
    while(1)
    {
        if (getchar())
            break;
    }
    printf("fun() ends \n\n");
}

// The main program calls fun() and measures time taken by fun()
int main()
{
    // Calculate the time taken by fun()
    clock_t t;
    t = clock();
    fun();
    t = clock() - t;
    double clocksPerSec = CLOCKS_PER_SEC;
```

```

double clocksSpent = t;

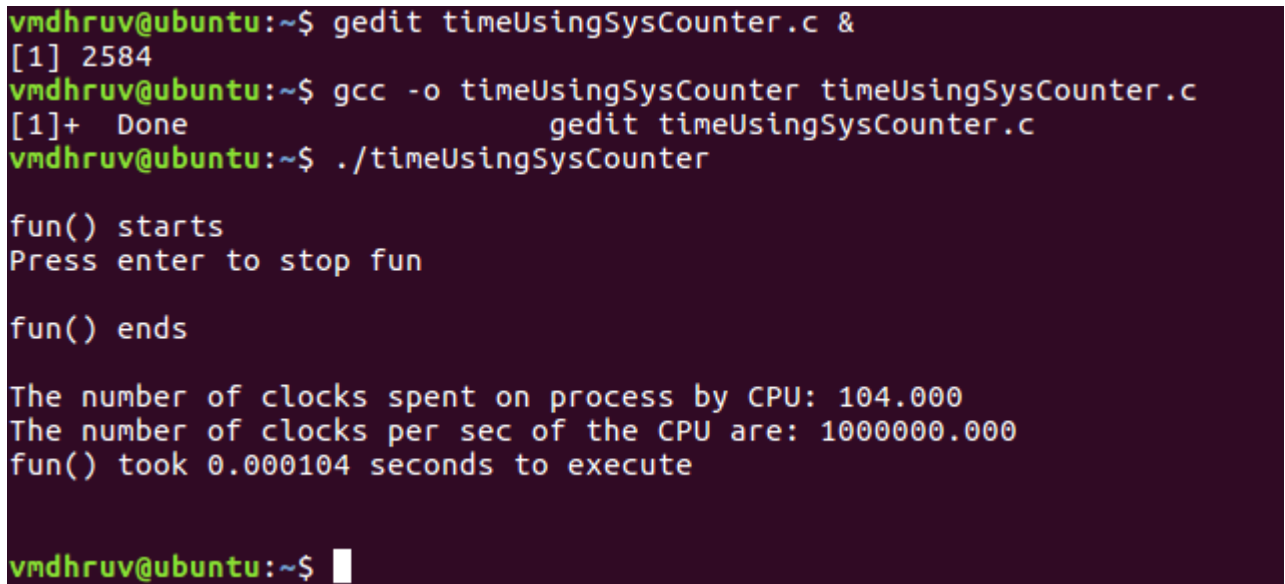
double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

printf("The number of clocks spent on process by CPU: %.3f\n", clocksSpent);
printf("The number of clocks per sec of the CPU are: %.3f\n", clocksPerSec);
printf("fun() took %f seconds to execute \n\n\n", time_taken);

return 0;
}

```

SCREENSHOT



```

vmdhruv@ubuntu:~$ gedit timeUsingSysCounter.c &
[1] 2584
vmdhruv@ubuntu:~$ gcc -o timeUsingSysCounter timeUsingSysCounter.c
[1]+  Done                  gedit timeUsingSysCounter.c
vmdhruv@ubuntu:~$ ./timeUsingSysCounter

fun() starts
Press enter to stop fun

fun() ends

The number of clocks spent on process by CPU: 104.000
The number of clocks per sec of the CPU are: 1000000.000
fun() took 0.000104 seconds to execute

vmdhruv@ubuntu:~$ █

```

INFERENCE

Here we can see that the frequency of the CPU is 1000000 clocks per second or 1 MHz frequency. Using the clocks used by the function, and dividing it by the clocks per second of the CPU, we can find the time taken by the process in seconds.