VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Lab 4:
System and Procedure Call

Programme	:	**BTech. CSE Core**	Semester	:	**Win 2021-22**
Course	:	**Operating Systems**	Code	:	**CSE2005**
Faculty	:	**Dr. Shyamala L**	Slot	:	**L25+L26**
Name	:	**Hariket Sukesh Kumar Sheth**	Register No.	:	**20BCE1975**

LAB 4

1. **Aim:** Write a simple test program to compare the cost of a simple procedure call to a simple system call (getpid(), fork() etc.,). To prevent the optimizing compiler from “optimizing out” your procedure calls, do not compile with optimization on. You should use a system call such as the UNIX gettimeofday() for time measurements. Design your code such that the measurement overhead is negligible. Also, be aware that timer values in some systems have limited resolution (e.g., millisecond resolution). Explain the difference (if any) between the time required by your simple procedure call and simple system call by discussing what work each call must do.

   To use gettimeofday:

```
struct timeval t1, t2;
double elapsedTime;
// start timer
gettimeofday(&t1, NULL);
// do something
// ...
// stop timer
gettimeofday(&t2, NULL);
// compute and print the elapsed time in millisec
elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms
fprintf(stderr, "elapsedTime = %5.3fms \n", elapsedTime);
```

Open ques1.c ~/Desktop/lab4 Save

```c
#include <sys/time.h>
#include <unistd.h>
#include <assert.h>
#include<stdio.h>
int foo()
{
        int A=4,B=20;
        A+=B;

        int i,fact=1,number=6;
        for(i=1;i<=number;i++)
        {
                fact=fact*i;
        }
        return 0;
}
long nanosec(struct timeval t)
{
        return((t.tv_sec*1000000+t.tv_usec)*1000);
}
int main()
{
        int j,res;
        long N_iterations=1000000000;
        float avgTimeSysCall, avgTimeFuncCall;
        struct timeval t1, t2;
        res=gettimeofday(&t1,NULL); assert(res==0);
        for (int 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 (int 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("Average time for System call getpid : %f\n",avgTimeSysCall);
        printf("Average time for Function call : %f\n",avgTimeFuncCall);
}
```

```c
#include <sys/time.h>
#include <unistd.h>
#include <assert.h>
#include<stdio.h>
int foo()
{
    int A=4,B=20;
    A+=B;

    int i,fact=1,number=6;
    for(i=1;i<=number;i++)
    {
        fact=fact*i;
    }
    return 0;
}
long nanosec(struct timeval t)
{
    return((t.tv_sec*1000000+t.tv_usec)*1000);
}
int main()
{
    int j,res;
    long N_iterations=1000000000;
    float avgTimeSysCall, avgTimeFuncCall;
    struct timeval t1, t2;
    res=gettimeofday(&t1,NULL); assert(res==0);
    for (int 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 (int 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("Average time for System call getpid : %f\n",avgTimeSysCall);
    printf("Average time for Function call : %f\n",avgTimeFuncCall);
}
```

```
hariketsheth@ubuntu: ~/Desktop/lab4
hariketsheth@ubuntu:~/Desktop/lab4$ gcc ques1.c -o ques1
hariketsheth@ubuntu:~/Desktop/lab4$ ./ques1
Average time for System call getpid : 64.286926
Average time for Function call : 8.755543
hariketsheth@ubuntu:~/Desktop/lab4$ gcc -o0 ques1.c -o ques1
hariketsheth@ubuntu:~/Desktop/lab4$ ./ques1

Average time for System call getpid : 66.672775
Average time for Function call : 8.663722
```

Solution: The cost of a system call is predicted to be much higher than the cost of a procedure call (provided that both perform very little actual computation). A system call comprises the following tasks that are not performed by a basic procedure call, resulting in a significant overhead:

a change of context

A trap in the interrupt vector at a given place

Control is sent to a service function in'monitor' mode.

What system call has happened is determined by the monitor.

The monitor double-checks that the parameters given are valid and proper.

2. Execution time calculation using Clock(). File contains words. Program to count the words.

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

void main() {
    clock_t begins = clock();
    int cp=0;
    FILE *fp;
    char ch;
    int wrd=0;
    fp = fopen("t1.txt","r");
    while ((ch = fgetc(fp)) != EOF)
    {
        if(ch==' '||ch=='\n')
         wrd++;
    }
    fclose(fp);
    printf("word count in file = %d",wrd);
    clock_t ends =clock();
    float timep = (float)(ends - begins)/ CLOCKS_PER_SEC;
    printf("\n Execution time taken for procedural call=%f\n", timep);
    begins = clock();
    system("wc -w t1.txt");
    ends = clock();
    float times = (float)(ends- begins) / CLOCKS_PER_SEC;
    printf("\n Execution time taken for system call=%f\n", times);
    printf("Time difference = %f\n", times-timep);
}
```

ques1.c ques2.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/syscall.h>
#include<time.h>
void main() {
    clock_t begins = clock();
    int cp=0;
    FILE *fp;
    char ch;
    int wrd=0;
    fp = fopen("t1.txt","r");
    while ((ch = fgetc(fp)) != EOF)
    {
        if(ch==' '||ch=='\n')
         wrd++;
    }
    fclose(fp);
    printf("word count in file = %d",wrd);
    clock_t ends =clock();
    float timep = (float)(ends - begins)/ CLOCKS_PER_SEC;
    printf("\n Execution time taken for procedural call=%f\n", timep);
    begins = clock();
    system("wc -w t1.txt");
    ends = clock();
    float times = (float)(ends- begins) / CLOCKS_PER_SEC;
    printf("\n Execution time taken for system call=%f\n", times);
    printf("Time difference = %f\n", times-timep);
}

```

3. Write a test program to compare the cost of a procedure call to a system call (getpid(), fork() etc.,). Design your code such that the measurement overhead is negligible. Also, be aware that timer values in some systems have limited resolution (e.g., millisecond resolution). Explain the difference (if any) between the time required by your simple procedure call and simple system call by discussing what work each call must do.

Procedure call to be used:

- Simple Addition\factorial\fibonacci\search\sort
- Any function containing some loop execution of your own.
- File contains some words. May be word count or searching name in it.

Execution types:

1) Run with compiler optimization enabled
   $gcc -O0 explore on this.

2) Run normally

```c
#include <sys/time.h>
#include <unistd.h>
#include <assert.h>
#include<stdio.h>
int foo()
{
    int A=4,B=20;
    A+=B;

    int i,fact=1,number=6;
    for(i=1;i<=number;i++)
    {
        fact=fact*i;
    }
    return 0;
}
long nanosec(struct timeval t)
{
    return((t.tv_sec*1000000+t.tv_usec)*1000);
}
int main()
```

Name: Hariket Sukesh Kumar Sheth Register No.: 20BCE1975

```c
{
    int j,res;
    double elapsedTime;
    long N_iterations=1000000000;
    float avgTimeSysCall, avgTimeFuncCall;
    struct timeval t1, t2;
    res=gettimeofday(&t1,NULL); assert(res==0);
    for (int 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 (int 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("Average time for System call getpid : %f\n",avgTimeSysCall);
    printf("Average time for Function call : %f\n",avgTimeFuncCall);
    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
    elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms
    fprintf(stderr, "elapsedTime = %5.3fms \n", elapsedTime);
}
```