Name – Ashutosh Ardu

Regno – 20BRS1262

OS LAB 5

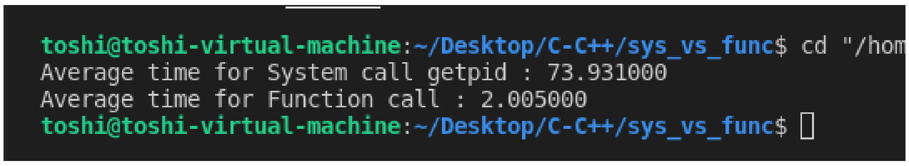
CODE

```c
#include <sys/time.h>
#include <unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include <assert.h>
int fibo(int n){
  if(n==0) return 0;
  else if(n==1 || n==2) return 1;
  else return fibo(n-1)+fibo(n-2);
}
int foo(){
  return 10;
}
int fact(int n){
  if(n==1 || n==0) return 1;
  else return n*fact(n-1);
}
long nanosec(struct timeval t){
  return((t.tv_sec*1000000+t.tv_usec)*1000);
}
int main(){
  int i,j,res;
  long N_iterations=50;
  float avgTimeSysCall, avgTimeFuncCall;
  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=fact(i);
    // j=foo();
    // j=fibo(i);
  }
  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);
}
```

OUTPUT

N=10^7

Running Getpid And Function Having Return 10

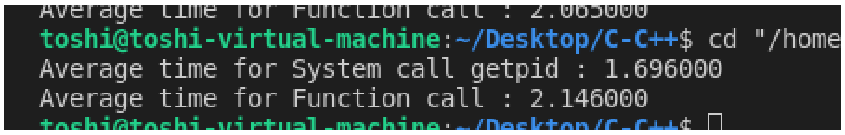


```
toshi@toshi-virtual-machine:~/Desktop/C-C++/sys_vs_func$ cd "/hom
Average time for System call getpid : 73.931000
Average time for Function call : 2.005000
toshi@toshi-virtual-machine:~/Desktop/C-C++/sys_vs_func$ 
```

(Naturally System Call Is More Expensive)

N=26*10^3

Running Getpid And Function Having Return 10

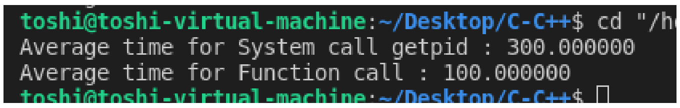


```
Average time for Function call : 2.065000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/home
Average time for System call getpid : 1.696000
Average time for Function call : 2.146000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(Almost Close)

N=10

Running Getpid And Fibonacco Function

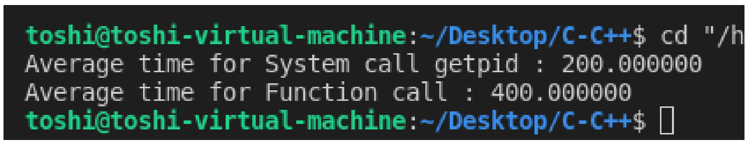


```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/h
Average time for System call getpid : 300.000000
Average time for Function call : 100.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(Still System Call Is More Expensive)

N=15

Running Getpid And Fibonacco Function

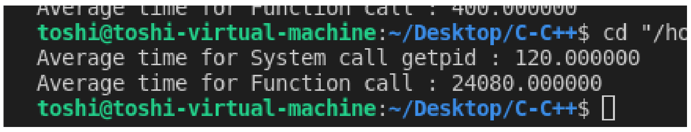


```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/h
Average time for System call getpid : 200.000000
Average time for Function call : 400.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(Here We Witness That For The First Time Function Call Is More Expensive Than System Call Under Similar Loop Condition)

N=25

Running Getpid And Fibonacco Function

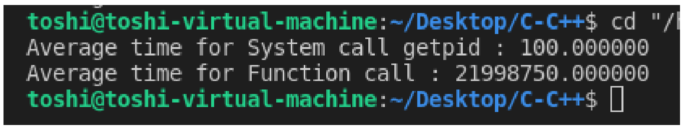


```
Average time for Function call : 400.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/ho
Average time for System call getpid : 120.000000
Average time for Function call : 24080.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(We Did Expect Function To Go Up Naturally (As Observed Before) But By Increasing N By Just 10 There Is An Exponential Increase In Expense For Function Calls)

N=40

Running Getpid And Fibonacco Function

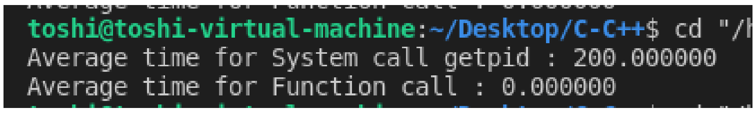


```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/
Average time for System call getpid : 100.000000
Average time for Function call : 21998750.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(Alas, Functional Call Has Becomes Far Too Expensive As Compared To System Call (As Expected From Previous Observations))

N=10

Running Getpid And Factorial Function

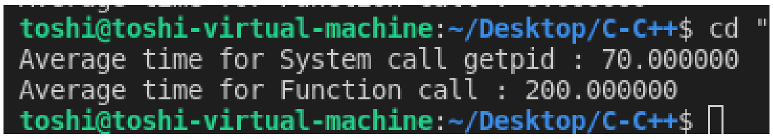


```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/h
Average time for System call getpid : 200.000000
Average time for Function call : 0.000000
```

(Here The Function Call Too Fast That Our Current Scale Are Not Good Enough To Capture It)

N=100

Running Getpid And Factorial Function

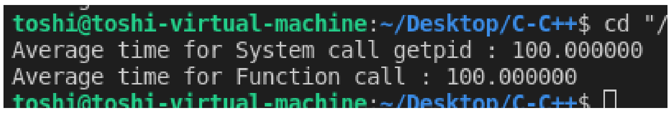


```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "
Average time for System call getpid : 70.000000
Average time for Function call : 200.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(Here As We Can See For The Again Function Call Is Becoming More Expensive Beyond A Certain Threshold Value Of N)

N=50

Running Getpid And Factorial Function



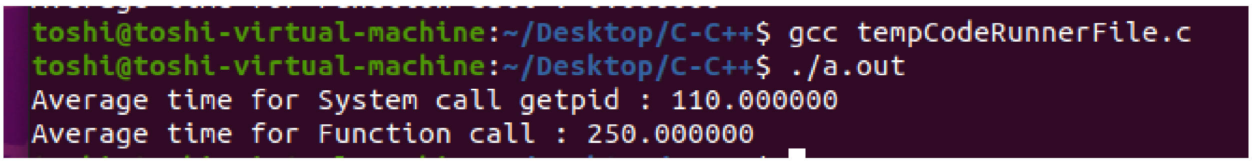
```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/
Average time for System call getpid : 100.000000
Average time for Function call : 100.000000
toshi@toshi-virtual-machine:~/Desktop/C-C++$ 
```

(The Function Call And System Approx. Similar Amount Of Time)

N=100

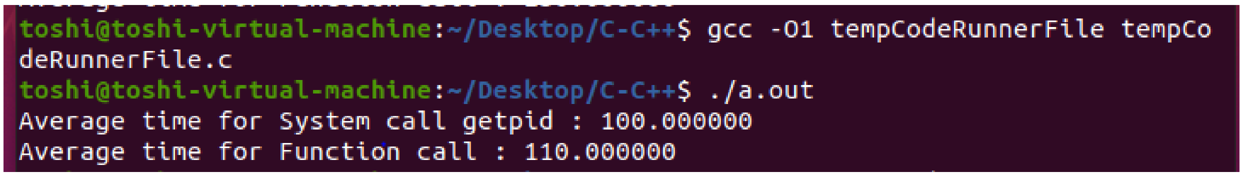
Running Getpid And Factorial Function

Under Normal Run



```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ gcc tempCodeRunnerFile.c
toshi@toshi-virtual-machine:~/Desktop/C-C++$ ./a.out
Average time for System call getpid : 110.000000
Average time for Function call : 250.000000
```

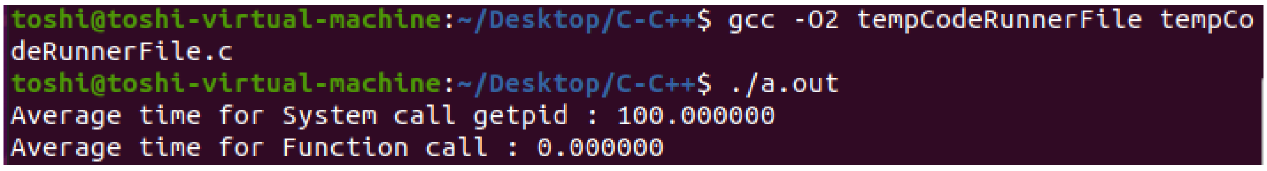
Now Using Optimization of Level 1



```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ gcc -O1 tempCodeRunnerFile tempCo
deRunnerFile.c
toshi@toshi-virtual-machine:~/Desktop/C-C++$ ./a.out
Average time for System call getpid : 100.000000
Average time for Function call : 110.000000
```

(As you can see the Complier Optimization has affected the time of the Functional Call in a drastic way making it almost equal to the time taken by System Call. But as we notice it didn't affect the time of System call at all.)

Now using Optimization of Level 2



```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ gcc -O2 tempCodeRunnerFile tempCo
deRunnerFile.c
toshi@toshi-virtual-machine:~/Desktop/C-C++$ ./a.out
Average time for System call getpid : 100.000000
Average time for Function call : 0.000000
```

(Now the Optimization has become so effective that the time that was initially taken by the Functional Call (which was more than the time take by the System Call by a large margin) has been drastically reduced. Almost making it so fast that our current time capturing scale could trace the time taken by it. But the optimization didn't affect the System Call at all.)

CONCLUSION

We conclude from the above mentioned observations that naturally the System calls takes much more time to complete its task as compared to functional calls. This is due to, context switching, determining specific location of interrupt vector, controlling passes to service routine and system call verificaiton by the kernel which is undertaken while calling a system call, these factors deteriorate the speed of execution as oppose to functional calls in a specific looping environment. But however with necessary complexiety in functional call and beyond threshold values of N(no of loops), the Functional call turns out to be more expensive as oppose to naturally expensive System call in a given looping environment. Also with the external addition of optimization to the compiler while compiling, affects the time taken for execution of the Functional Call without having any effect on the time taken by the System Call execution. Also the drop in the time taken by the Functional Call's execution is decided by the level optimization used. More the optimization level used less is the time taken by the Functional Call's execution.