CSE2005L - Operating Systems Lab

Lab 6 - Overhead in system and procedure call

PRANAVCHENDUR T K - 15BCE1097

Faculty : Dr. Shyamala.L

Code to Measure and print System Call

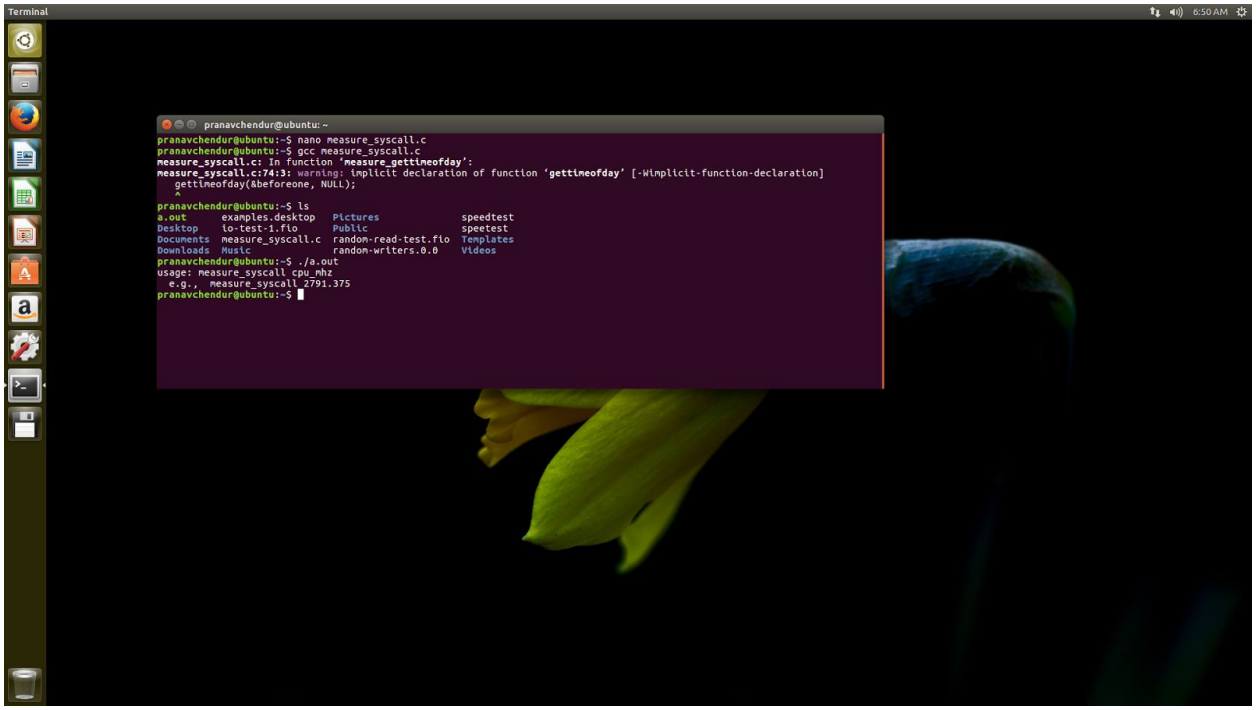
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/syscall.h>

// only works on pentium+ x86
// access the pentium cycle counter
// this routine lifted from somewhere on the Web...
void access_counter(unsigned int *hi, unsigned int *lo) {
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"   /* Read cycle counter */
      : "=r" (*hi), "=r" (*lo)                /* and move results to */
      : /* No input */                        /* the two outputs */
      : "%edx", "%eax");
}

// here's the system call we'll use
#define DO_SYSCALL syscall(SYS_getpid)

// calculate difference (in microseconds) between two struct timevals
// assumes difference is less than 2^32 seconds, and unsigned int is 32
bits
unsigned int timediff(struct timeval before, struct timeval after) {
  unsigned int diff;

  diff = after.tv_sec - before.tv_sec;
  diff *= 1000000;
  diff += (after.tv_usec - before.tv_usec);

  return diff;
}

// measure the system call using the cycle counter.  measures the
// difference in time between doing two system calls and doing
// one system call, to try to factor out any measurement overhead
```

```
void measure_cyclecounter(float mhz) {
  unsigned int high_s, low_s, high_e, low_e;
  size_t nbytes;
  float latency_with_read, latency_no_read;

  // warm up all the caches by exercising the functions
  access_counter(&high_s, &low_s);
  // read(5, buf, 4);
  DO_SYSCALL;
  access_counter(&high_e, &low_e);

  // now do it for real
  access_counter(&high_s, &low_s);
  DO_SYSCALL;
  access_counter(&high_e, &low_e);
  latency_with_read = ((float) (low_e - low_s) / mhz);

  access_counter(&high_s, &low_s);
  access_counter(&high_e, &low_e);
  latency_no_read = ((float) (low_e - low_s) / mhz);

  // print out the results
  printf("(cyclecounter)  latency:  %f microseconds\n", latency_with_read
- latency_no_read);
}

// measure the system call using the cycle counter.  measures the
// difference in time between doing two*NLOOPS system calls and doing
// one*NLOOPS system calls, to try to factor out any measurement overhead

#define NUMLOOPS 10000
void measure_gettimeofday() {
  struct timeval beforeone, afterone;
  struct timeval beforetwo, aftertwo;
  int loopcount;
  unsigned int diffone, difftwo, result;

  // warm up all caches
  gettimeofday(&beforeone, NULL);
  gettimeofday(&beforetwo, NULL);
  for (loopcount = 0; loopcount < NUMLOOPS; loopcount++) {
    DO_SYSCALL;
  }
  gettimeofday(&afterone, NULL);
```

```
  gettimeofday(&aftertwo, NULL);

  // measure loop of one syscall
  gettimeofday(&beforeone, NULL);
  for (loopcount = 0; loopcount < NUMLOOPS; loopcount++) {
    DO_SYSCALL;
  }
  gettimeofday(&afterone, NULL);

  // measure loop of two syscalls
  gettimeofday(&beforetwo, NULL);
  for (loopcount = 0; loopcount < NUMLOOPS; loopcount++) {
    DO_SYSCALL;
    DO_SYSCALL;
  }
  gettimeofday(&aftertwo, NULL);

  diffone = timediff(beforeone, afterone);
  difftwo = timediff(beforetwo, aftertwo);

  result = difftwo - diffone;
  printf("(loop)  latency:  %f microseconds\n", ((float) result) /
((float) NUMLOOPS));
}

void usage(void) {
  fprintf(stderr, "usage: measure_syscall cpu_mhz\n");
  fprintf(stderr, "  e.g.,  measure_syscall 2791.375\n");
  exit(-1);
}



int main(int argc, char **argv) {

  float mhz;

  if (argc < 2)
    usage();

  if (sscanf(argv[1], "%f", &mhz) != 1)
    usage();

  if ((mhz < 100.0)  || (mhz > 100000.0))
    usage();
```

```
  // measure usiing the cycle counter
  measure_cyclecounter(mhz);

  // measure using "gettimeofday"
  measure_gettimeofday();

  return 0;
}
```

Output

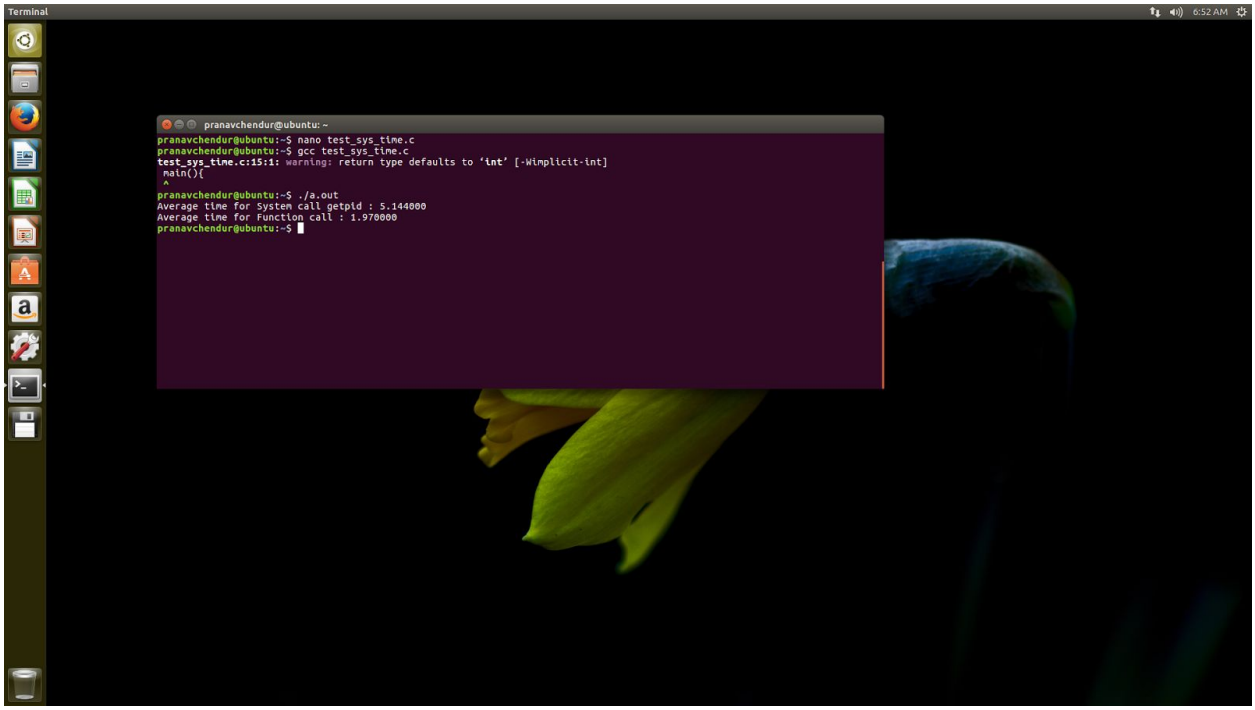


Code to measure time of System Call Execution

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


int foo(){
  return(10);
```

```
}

long nanosec(struct timeval t){ /* Calculate nanoseconds in a timeval
structure */
  return((t.tv_sec*1000000+t.tv_usec)*1000);
}

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

  /* Find average time for System call */
  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);

  /* Find average time for Function call */
  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("Average time for System call getpid : %f\n",avgTimeSysCall);
  printf("Average time for Function call : %f\n",avgTimeFuncCall);
}
```

Output:


Terminal
6:52 AM
pranavchendur@ubuntu: ~
pranavchendur@ubuntu:~$ nano test_sys_time.c
pranavchendur@ubuntu:~$ gcc test_sys_time.c
test_sys_time.c:15:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(){
^
pranavchendur@ubuntu:~$ ./a.out
Average time for System call getpid : 5.144000
Average time for Function call : 1.970000
pranavchendur@ubuntu:~$