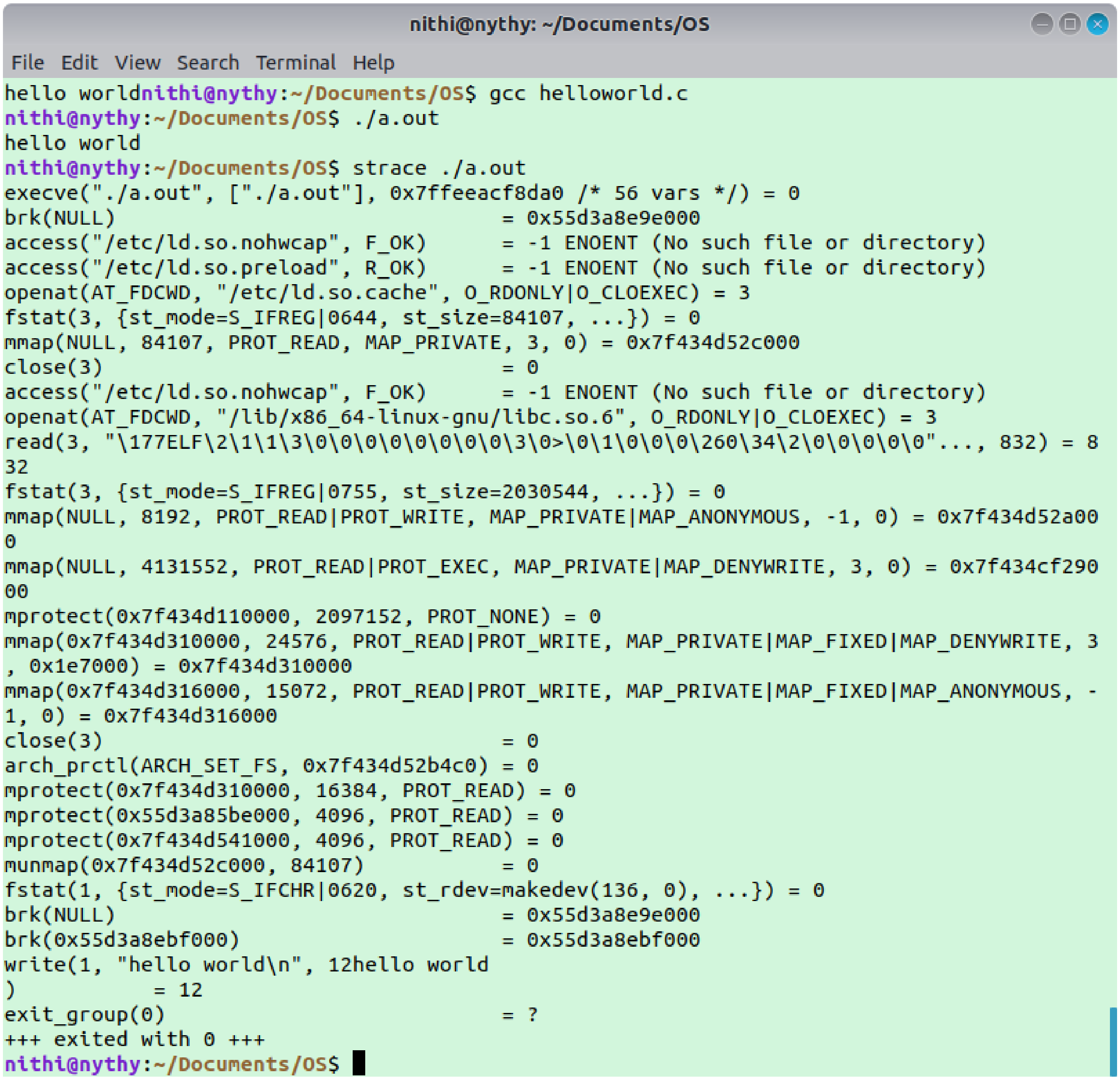
Nithilam Subbaian
Operating Systems
ECE-357

# PSET 7

## Problem 1 -- using strace

```
#include <stdio.h>
int main()
{
   printf("hello world\n");
   return 0;
}
```

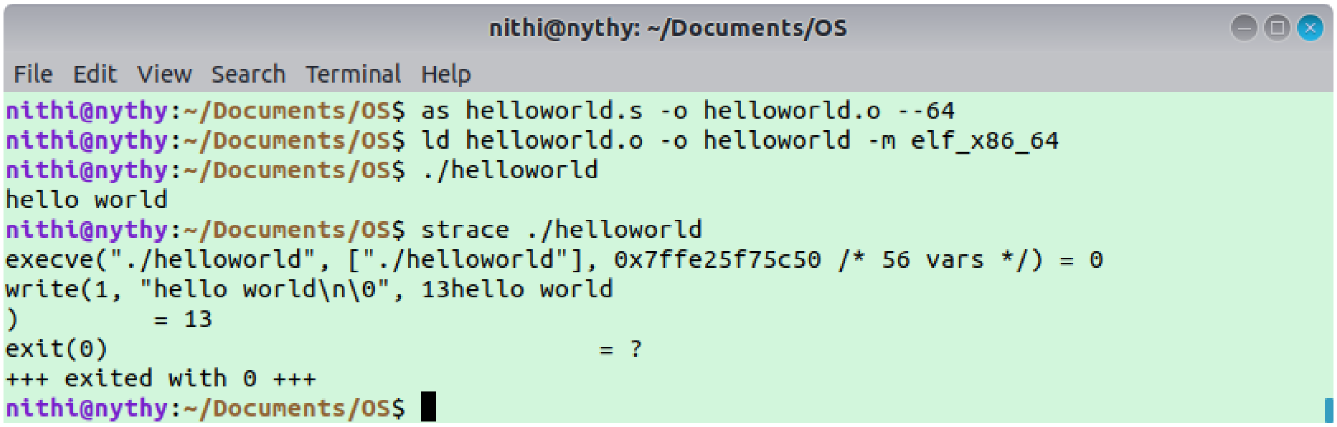

nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
hello worldnithi@nythy:~/Documents/OS$ gcc helloworld.c
nithi@nythy:~/Documents/OS$ ./a.out
hello world
nithi@nythy:~/Documents/OS$ strace ./a.out
execve("./a.out", ["./a.out"], 0x7ffeeacf8da0 /* 56 vars */) = 0
brk(NULL)                               = 0x55d3a8e9e000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=84107, ...}) = 0
mmap(NULL, 84107, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f434d52c000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 8
32
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f434d52a00
0
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f434cf290
00
mprotect(0x7f434d110000, 2097152, PROT_NONE) = 0
mmap(0x7f434d310000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3
, 0x1e7000) = 0x7f434d310000
mmap(0x7f434d316000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f434d316000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f434d52b4c0) = 0
mprotect(0x7f434d310000, 16384, PROT_READ) = 0
mprotect(0x55d3a85be000, 4096, PROT_READ) = 0
mprotect(0x7f434d541000, 4096, PROT_READ) = 0
munmap(0x7f434d52c000, 84107)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
brk(NULL)                               = 0x55d3a8e9e000
brk(0x55d3a8ebf000)                     = 0x55d3a8ebf000
write(1, "hello world\n", 12hello world
)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
nithi@nythy:~/Documents/OS$ 
```

# Problem 2 -- pure assembly

**Source Code:**

```
.text

.global _start

_start:
     mov      $1, %rax
     mov      $1, %rdi
     mov      $message, %rsi
     mov      $13, %rdx
     syscall

     mov      $60, %rax
     xor      %rdi, %rdi
     syscall
     message:
     .ascii   "hello world\n"
```


nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
nithi@nythy:~/Documents/OS$ as helloworld.s -o helloworld.o --64
nithi@nythy:~/Documents/OS$ ld helloworld.o -o helloworld -m elf_x86_64
nithi@nythy:~/Documents/OS$ ./helloworld
hello world
nithi@nythy:~/Documents/OS$ strace ./helloworld
execve("./helloworld", ["./helloworld"], 0x7ffe25f75c50 /* 56 vars */) = 0
write(1, "hello world\n\0", 13hello world
)         = 13
exit(0)                                 = ?
+++ exited with 0 +++
nithi@nythy:~/Documents/OS$
```

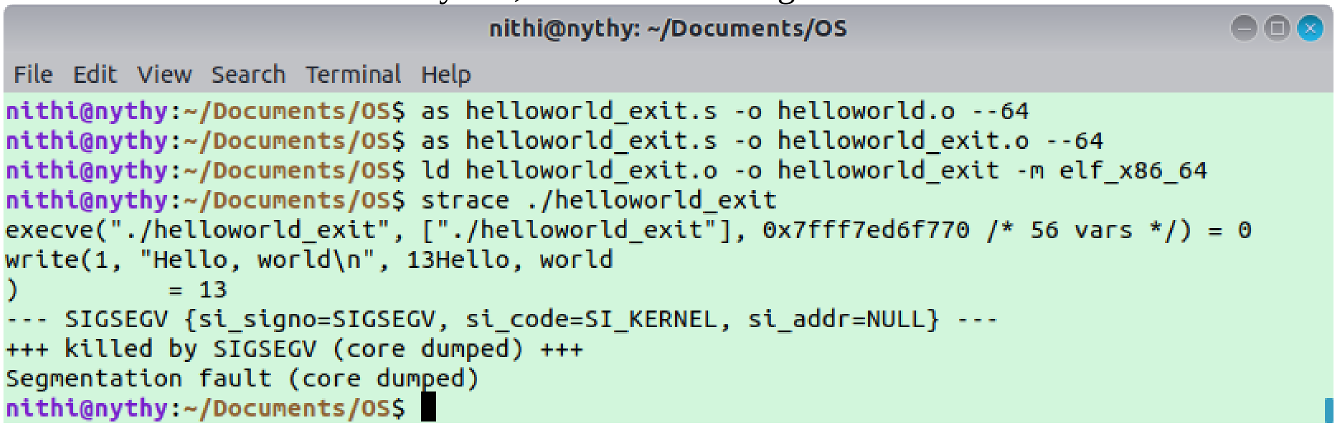
## Problem 3 -- exit code

From observing the strace output after the write system call from the code above, there is a line “exit(0)”. This comes from the lines “mov    $60, %rax”, which selects system call 60, the exit system call. The line “xor        %rdi, %rdi”, sets the return value to 0. Then the line “syscall” invokes the operating system to exit.

If the lines :

```
    mov     $60, %rax
    xor     %rdi, %rdi
    syscall
```

are removed from the assembly file, then the following result occurs:


nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
nithi@nythy:~/Documents/OS$ as helloworld_exit.s -o helloworld.o --64
nithi@nythy:~/Documents/OS$ as helloworld_exit.s -o helloworld_exit.o --64
nithi@nythy:~/Documents/OS$ ld helloworld_exit.o -o helloworld_exit -m elf_x86_64
nithi@nythy:~/Documents/OS$ strace ./helloworld_exit
execve("./helloworld_exit", ["./helloworld_exit"], 0x7fff7ed6f770 /* 56 vars */) = 0
write(1, "Hello, world\n", 13Hello, world
)           = 13
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=NULL} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
nithi@nythy:~/Documents/OS$ 
```

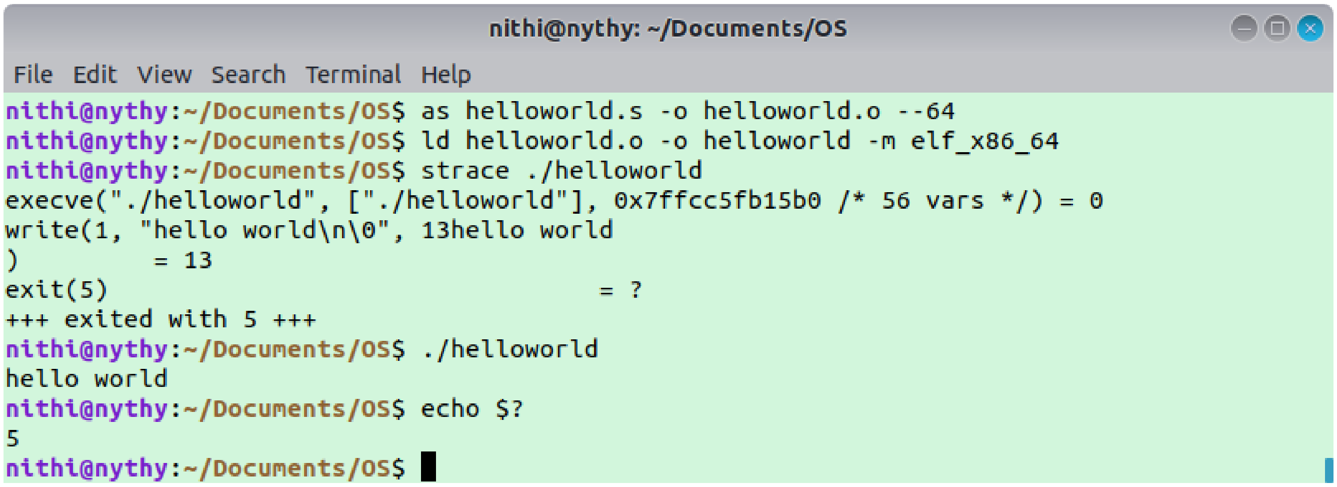
Here, after the write system call, the kernel returns to userspace but as  the text region has nothing, the program continues to try and run unknown unknown random memory and the processor will attempt to execute the values in this memory as instructions. As this memory is unknown, these bytes will likely not be valid instructions, leading to a fault, or cause an error, such as entering an endless loop.

Another version of the program that has the _exit system call so that the program exits with a specific non-zero return code:

Source Code:

```
.text

.global _start

_start:
    mov     $1, %rax
    mov     $1, %rdi
    mov      $message, %rsi
    mov      $13, %rdx
    syscall

    mov      $60, %rax
        mov $5, %rdi
    syscall
    message:
    .ascii   "hello world\n"
```

Here the exit code is set to 5, and so echo $? returns a 5 as shown bellow in the screenshot:


nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
nithi@nythy:~/Documents/OS$ as helloworld.s -o helloworld.o --64
nithi@nythy:~/Documents/OS$ ld helloworld.o -o helloworld -m elf_x86_64
nithi@nythy:~/Documents/OS$ strace ./helloworld
execve("./helloworld", ["./helloworld"], 0x7ffcc5fb15b0 /* 56 vars */) = 0
write(1, "hello world\n\0", 13hello world
)          = 13
exit(5)                                 = ?
+++ exited with 5 +++
nithi@nythy:~/Documents/OS$ ./helloworld
hello world
nithi@nythy:~/Documents/OS$ echo $?
5
nithi@nythy:~/Documents/OS$ 
```

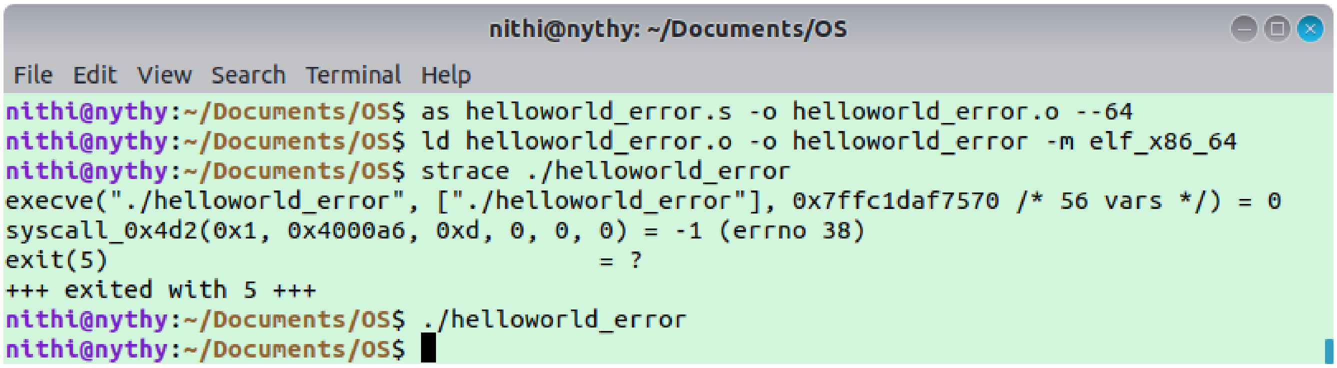
## Problem 4 -- system call validation

Here is a version of the part 3 program that passes an invalid system call number to write. Shown bellow is the code and the strace output. The system call for

1 is write, but here that number of 1 is changed. And therefore, the write system call is not called, but the program exits, as that code is still there.

Source Code:

```
.global _start

.text

_start:
         mov     $1234, %rax
         mov     $1, %rdi
         mov     $message, %rsi
         mov     $13, %rdx
         syscall

         # exit(5)
         mov     $60, %rax
         mov     $5, %rdi
         syscall

         message:
.        ascii "hello world\n"
```


nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
nithi@nythy:~/Documents/OS$ as helloworld_error.s -o helloworld_error.o --64
nithi@nythy:~/Documents/OS$ ld helloworld_error.o -o helloworld_error -m elf_x86_64
nithi@nythy:~/Documents/OS$ strace ./helloworld_error
execve("./helloworld_error", ["./helloworld_error"], 0x7ffc1daf7570 /* 56 vars */) = 0
syscall_0x4d2(0x1, 0x4000a6, 0xd, 0, 0, 0) = -1 (errno 38)
exit(5)                                 = ?
+++ exited with 5 +++
nithi@nythy:~/Documents/OS$ ./helloworld_error
nithi@nythy:~/Documents/OS$ 
```

The error 38  /* Function not implemented */ comes up because that system call does not exist.

## Problem 5EC -- scheduling

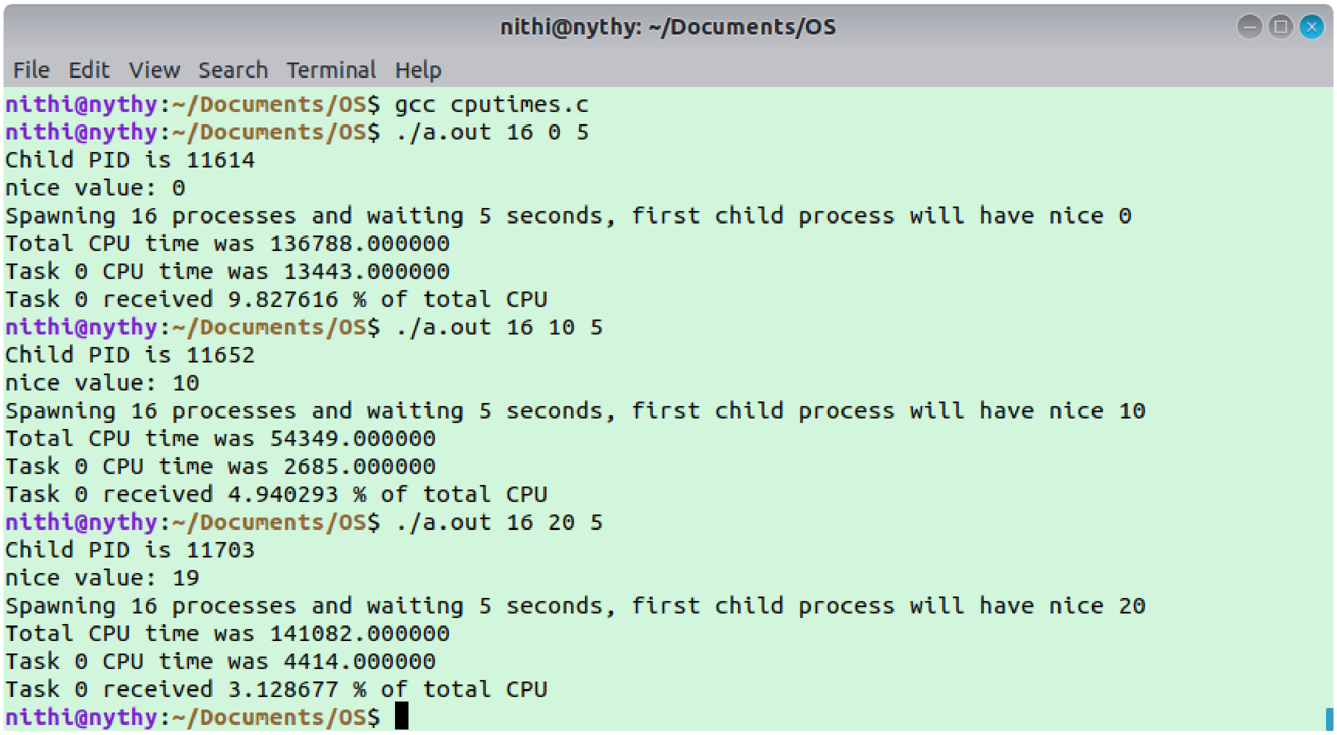
Write a test program which creates a number of CPU-bound processes.

One of those processes will have a non-default `nice` value. Both the number of processes and the nice value will be command-line flags. After spawning the processes, the parent process will sleep for the specified number of seconds, then kill all the children and collect their respective `rusage` structures. Report the sum total CPU time consumed by the children, and the CPU time by the child which had the non-default nice value. Note: include both system and user CPU time.

Source Code:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/times.h>
#include <time.h>

int i, j;
int main(int argc, char *argv[]){
        int processCount =atoi(argv[1]);
        int waitTime = atoi(argv[3]);
        int niceNumber =  atoi(argv[2]);

        int count = processCount-1;
        pid_t pid;
        pid_t pidtask0;

        int ret;
        struct rusage usagetask0, usageallchild;

        pid_t cpid, w, w2;
        int status;

        cpid = fork();
        if (cpid == -1) { perror("ERROR: fork failed");
exit(EXIT_FAILURE); }

        if (cpid == 0) {
                printf("Child PID is %ld\n", (long)
getpid());
                ret  = nice(niceNumber);
                printf("nice value: %d\n",ret);
                for (i=0, j=0; i<1000000; i++) {
                        for (i=0, j=0; i<1000000; i++)
                                j += i;
                }
                if (argc == 1)
                        pause();
                exit(0);
        }


        int count2=count;
        while(count--) {
                if((pid=fork())<0) {
                        printf("ERROR: fork error\n");
                } else if(pid==0) {
                        int i;
                        for (i=0, j=0; i<1000000; i++) {
                                for (i=0, j=0; i<1000000; i+
+)
                                        j += i;
                        }
                        exit(0);
                }
        }

        sleep(waitTime);

        do {
                w = waitpid(cpid, &status, WUNTRACED |
WCONTINUED);
                getrusage (RUSAGE_CHILDREN, &usagetask0);
                if (w == -1) { perror("ERROR: waitpid for
task with change in nice value"); exit(EXIT_FAILURE); }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));

        count2;
        while(count2--) {
                w2 = wait(NULL);
                if (w2 == -1) { perror("ERROR: wait on
children failed"); exit(EXIT_FAILURE); }

        }

        getrusage (RUSAGE_CHILDREN, &usageallchild);
        printf("Spawning %d processes and waiting %d seconds,
first child process will have nice %d\n", processCount,
waitTime, niceNumber);

        double childtime = usageallchild.ru_utime.tv_sec +
usageallchild.ru_utime.tv_usec +
usageallchild.ru_stime.tv_sec +
usageallchild.ru_stime.tv_usec;
        double task0time =   usagetask0.ru_utime.tv_sec +
usagetask0.ru_utime.tv_usec + usagetask0.ru_stime.tv_sec +
usagetask0.ru_stime.tv_usec;

        printf("Total CPU time was %lf\n", childtime +
task0time );
        printf("Task 0 CPU time was %lf \n", task0time);
        printf("Task 0 received %f %% of total CPU\n",
(task0time)/(childtime +task0time)*100);

        return 0;
}
```

**Sample Output:**



nithi@nythy: ~/Documents/OS
File Edit View Search Terminal Help

```
nithi@nythy:~/Documents/OS$ gcc cputimes.c
nithi@nythy:~/Documents/OS$ ./a.out 16 0 5
Child PID is 11614
nice value: 0
Spawning 16 processes and waiting 5 seconds, first child process will have nice 0
Total CPU time was 136788.000000
Task 0 CPU time was 13443.000000
Task 0 received 9.827616 % of total CPU
nithi@nythy:~/Documents/OS$ ./a.out 16 10 5
Child PID is 11652
nice value: 10
Spawning 16 processes and waiting 5 seconds, first child process will have nice 10
Total CPU time was 54349.000000
Task 0 CPU time was 2685.000000
Task 0 received 4.940293 % of total CPU
nithi@nythy:~/Documents/OS$ ./a.out 16 20 5
Child PID is 11703
nice value: 19
Spawning 16 processes and waiting 5 seconds, first child process will have nice 20
Total CPU time was 141082.000000
Task 0 CPU time was 4414.000000
Task 0 received 3.128677 % of total CPU
nithi@nythy:~/Documents/OS$ 
```

## Table With Changed Values:

### Average % of total CPU that task 0 received for Parent Sleeping for 5 seconds for 3 runs each:

| | Nice = 0 | Nice = 10 | Nice = 19 |
|---|---|---|---|
| ProcessCount = 3 | 26.666772 | 25.756703 | 25.770789 |
| ProcessCount = 5 | 17.23830 | 15.12561 | 14.42673 |
| ProcessCount = 16 | 9.827616 | 4.940293 | 3.128677 |
| ProcessCount = 35 | 4.752382 | 2.708152 | 2.110145 |
| ProcessCount = 100 | 2.552680 | 1.676030 | 0.974388 |

I ran the above on a CPU with 4 cores. So using the same logic in the program description, over a 5-second period, the maximum theoretical CPU time would be 20 seconds. The 16 test processes did not get a number that was near this value, only leading me to presume that the system was otherwise not idle, even though I don't have much else running.

The sleep time was set to 5 seconds as because as noted in the program description, that's a minimum so that otherwise brief fluctuations in system load don't drastically affect your answers.

This was then run for various combinations of processes and nice values. Smaller than the number of cores is 3, so the program was run for 3 processes, and the large number of 100 was chosen to see the other end.  The nice value appears to not affect things for small numbers of processes relative to CPU cores. This makes sense because there is less of a need to schedule or prioritize processes because all processes can be computed with the relative process count to the core count.