Computer Science and Engineering & Information Technology (2^{nd} Year B.Tech.) IIIT Kalyani, West Bengal

Operating System Lab (CS 411): (Spring: 2019-2020)

Assignment - 2 *Marks: 20*

Assignment Out: *24th January, 2020*

1. We have already seen that the system call

```
int pid;
pid = fork();
```

can be replaced by the *inline assembly code* (x86-64) as follows:

```
  __asm__ __volatile__(
       "movq $57,%%rax \n\t"
       "syscall \n\t"
       :"=a" (pid)
   ) ;
```

The command for `fork()` is 57, loaded in the register `rax`, `syscall` is the machine instruction for *software interrupt* `trap`, the return value of `fork()` is available in `eax` (`rax[0-31]`), it goes to C++ variable `pid`.

Consider the following C++ program with the system call `read()` (see the man page).

```
#include <iostream>
using namespace std;
#include <unistd.h>
#define MAXL 201

int main(){
    char data[MAXL]={0};
    int n, bytes;

    cout << "Enter a positive integer <= " << MAXL-1 << endl;
    cin >> n;
    cout << "Enter a string\n";
    bytes=read(STDIN_FILENO, data, n);
    cout << data << endl;
    cout << "Bytes read: " << bytes << endl;
    return 0;
}
```

A run of the program:

```
$ a.out
Enter a positive integer <= 200
10
Enter a string
IIIT Kalyani
IIIT Kalya
Bytes read: 10
$ ni
ni: command not found
```

Write a C++ program where you replace the call `bytes=read(0, data, n);` by inline assembly language code of x86-64. Note the following information.

(a) The command for `read` is 0 (zero). It goes to register `rax`.

(b) The first parameter is the *file descriptor* for the input file. For `stdin` it is 0 (zero) (symbolic name `STDIN_FILENO` defined in `unistd.h`). The first parameter goes to register `rdi`.

(c) The second parameter is the starting address of the buffer `data`. See inline assembly language manual to map this name of C++ variable to the second parameter register `rsi` (code ”S”).

(d) The third parameter is the number of bytes to read. This is passed through the register `rdx` (code ”d”). Again see the manual to map C++ variable `n` to this CPU register.

(e) The return value is available in the CPU register `eax` (`rax[0-31]`). The C++ variable `bytes` is mapped to it for output.

2. Write a Python program that reads a non-negative integer n and creates n child processes. Each child process prints its process ID. A sample run looks like:

```
$ nProc.py
Enter a non-negative integer: 4
child: 6616
child: 6617
child: 6618
child: 6615
```

Use `os._exit(os.EX_OK)` to terminate a process.

3. Write a C++ program that reads a non-negative integer n and creates $n!$ child processes. **You are not allowed to pre-compute the value of** $n!$ i.e. given the value of $n = 4$ you **cannot** compute $4! = 24$ first and then create 24 processes. Each child process prints its process ID.

The **structure** of the following recursive function to compute $n!$ may help you to solve the problem! Note that to terminate a process from a function `exit(1)` can be used.

```
int factorial(int n){
    int val = 0;
    if(n==0) return 1;
    for(int i=1; i<=n; ++i)
       val += factorial(n-1);
    return val;
}
```

A sample run:

```
$ a.out
Enter a positive integer: 3
child: 6667
child: 6668
child: 6669
child: 6671
child: 6666
child: 6670
```