National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS 330: Operating Systems

BSCS: 12 C

Lab 4: Processes

Date: 28/2/2023

Time: 10:00 AM-01:00 PM

Instructor: Dr. Muhammad Zeeshan

Lab 4 : Processes

Introduction

Process s an instance of a program in execution. A process provides the illusion that the program has exclusive use of the processor and exclusive use of the memory system

Objectives

Understand the processes

Tools/Software Requirement

Gcc/G++ Linux

Description

Process

Is an instance of a program in execution. A process provides the illusion that the program has exclusive use of the processor and exclusive use of the memory system

fork()

When the fork system call is executed, a new process is created which consists of a copy of the address space of the parent.

- **Negative Value**: creation of a child process was unsuccessful.
- **Zero**: Returned to the newly created child process.
- **Positive value**: Returned to parent or caller. The value contains process ID of newly created child process.

Code-1

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h> //: Defines pid_t definition.
#include <unistd.h> //Defines System Calls including fork()
 int main(int argc, char *argv[])
{
pid_t pid; //return type of process ID
pid = fork();
   if (pid < 0)
  {
      // fork failed
      printf("fork failed\n");
      return 0;
   }
   else if (pid == 0)
   {
      // child (new process)
   }
   else
   {
      // parent goes down this path (original process)
   }
   return 0; }
```

wait()

The wait system call suspends the calling process until one of its immediate children terminates.

If the call is successful, the process ID of the terminating child is returned.

<sys/wait.h> ios used to defines wait System Call.

Zombie process

A process that has terminated but whose exit status has not yet been received by its parent process.

- the process will remain in the operating system's process table as a zombie process, indicating that it is not to be scheduled for further execution
- But that it cannot be completely removed (and its process ID cannot be reused)

*pid_t wait(int *status);*

Where status is an integer value where the UNIX system stores the value returned by child process

Code-2:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h> //: Defines pid_t definition.
#include <unistd.h> //Defines System Calls including fork()
void main()
{
pid_t pid, status;
 pid = fork();
 if(pid == -1)
{
 printf("fork failed\n");
```

```
exit(1);
}
if(pid == 0)
{ /* Child */
printf("Child here!\n");
}
else
{ /* Parent */
wait(&status);
printf("Well done kid!\n");
}
}
```

getpid()

The getpid() function returns the pid of the process which calls it.

Tasks

Its an individual lab

Task # 1

Reuse your code of a simple calculator (from the previous lab) and add the following functionalities into it.

1. To find the factorial of a natural number.

Your code should take a natural number as input from the user and then print the value of its

factorial. Hint: use loops in shell.

2. To check if a number is even or odd.

Your code should take a number and check whether it is odd or even. It should print the result accordingly.

3. Find the square root of a natural number.

Your code should take a natural number as input from the user and then print its square root.

4. Find the square and cube of a real number

Your code should take a real number as input from the user and then print its square and cube value.

The initial version of the calculator which you programmed in the previous lab had the capability to perform four fundamental arithmetic operations, now it should be capable of performing eight operations.

Task # 2

1. **Run the following code and find the number of time printf is executed. What do we interpret from it**

Sample:1

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // make two process which run same

    // program after this instruction

    fork();

    printf("Hello world!\n");

    return 0;

}
```

Sample:2

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("Hello world!\n");
    return 0;
}
```

Sample:3

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    fork();
    printf("Hello world!\n");
    return 0;
}
```

Sample:4

```
#include <stdio.h>
```

```
#include <sys/types.h>

int main()

{

Int n=10;

for(int i=0; i<n; i++){

    fork();

    printf("Hello world!\n");

    }

    return 0;

}
```

2. Predict the output of the following code. Why it differs in every execution.

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // child process because return value zero

    if (fork() == 0)

            printf("Hello from Child!\n");

    // parent process because return value non-zero.

    else

            printf("Hello from Parent!\n");

    return 0;

}
```

3. **Use getpid() to print the process ids of both parent and child for the code in task 2.**

Deliverables

Submit the document containing the code and output of the tasks.