Operating System- CS604

Solution Assignment # 1

Spring 2011

Marks: 20

Due Date

Your assignment must be uploaded before or on **April 18, 2011**

Objective

The objective of this assignment is to familiarize with the system calls.

Instructions

- Avoid Plagiarism. No marks will be given in case of cheating or copying from the internet or from other students.
- Submit the assignment through your account on VULMS. No assignment will be accepted through email after the due date.
- If you have any problem related to assignment, feel free to discuss it by email at cs604@vu.edu.pk

Question # 1:

Read the following program carefully and write the output of the program. Explain each line of code according to given numbering.

Output:

I have no child: 0
I AM VU: 0
I have no child: 1
I AM VU: 1
I have no child: 2
I AM VU: 2
I have no child: 3
I AM VU: 3
I have no child: 4
I AM VU: 4

Comment:

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

```
#include <stdlib.h>
#include <errno.h>

1…… int main (void)
```

The main function starts the program execution and returns int data type.

```
        {
                pid_t  pid;
2……             pid = fork();
```

Fork ( ) method is the system call and returns the integer value in the pid variable. Generate a clone of the existing process.

```
3……    if (pid > 0)
```

Condition will be only true when fork returned the vale greater than zero. Means fork is successful, a new process has been generated and parent process execution starts.

```
                int i;
4……      for (i = 0; i < 5; i++)
```

Loop starts and from 0 to 4, loop run 5 times.

```
                {
5……         printf("I AM VU: %d\n", i);
```

Prints “I AM VU” message on the screen and also print the value of variable “i”.

```
6……        sleep(1);
```

Sleep function suspend the execution for one second each time.

```
                }
                exit(0);
        }
7……    else if (pid == 0)
```

When fork () returns 0 in child process. The execution of the child process starts.

```
        {
                int j;
                        for (j = 0; j < 5; j++)
                {
8……         printf("I have no child: %d\n", j);
```

A message “I have no child” is printed on the screen and also print the value of variable “j”.

```
                sleep(1);
        }
                        _exit(0);
                }
                        else
         {
9……..          fprintf(stderr, "can't fork, error %d\n", errno);
```

This means fork has failed , (due to standard error ,so it has returned -1. it print message “can't fork, error” and print the error number.

```
10……..          exit (EXIT_FAILURE);
```

System call terminates the process abnormally as it fails. Exit function indicates unsuccessful program completion. Using the macro Exit_Failure

```
                }
        }
```

```
#include <stdio.h>   /* printf, stderr, fprintf */
#include <unistd.h>  /* _exit, fork */
#include <stdlib.h>  /* exit */
#include <errno.h>   /* errno */

int main(void)
{
   pid_t  pid;

   /* Output from both the child and the parent process
    * will be written to the standard output,
    * as they both run at the same time.
    */
   pid = fork();
   if (pid == 0)
   {
      /* Child process:
       * When fork() returns 0, we are in
       * the child process.
       * Here we count up to ten, one each second.
       */
      int j;
      for (j = 0; j < 10; j++)
      {
         printf("child: %d\n", j);
         sleep(1);
      }
      _exit(0);  /* Note that we do not use exit() */
   }
   else if (pid > 0)
   {
      /* Parent process:
       * When fork() returns a positive number, we are in the parent
process
       * (the fork return value is the PID of the newly-created child
process).
       * Again we count up to ten.
       */
      int i;
      for (i = 0; i < 10; i++)
      {
         printf("parent: %d\n", i);
         sleep(1);
      }
      exit(0);
   }
   else
   {
      /* Error:
       * When fork() returns a negative number, an error happened
       * (for example, number of processes reached the limit).
       */
      fprintf(stderr, "can't fork, error %d\n", errno);
      exit(EXIT_FAILURE);
   }
}
```