ICS 2305 systems Programming

Lab Exercises on Process Management – 08/10/2024

Using C programming

1. Creating a new process using fork() system call
2. CREATING multiple processes using fork () and pipe()
3. Demonstrate the concept of Orphan Process.
4. **Using The exec Family of Calls---- execlp(),execle(),execv(),execvp(),exeve()**

Practice cases

1. **This program prints the process PID then exits**

```
#include <stdlib.h>  /* needed to define exit() */
#include <unistd.h>  /* needed to define getpid() */
#include <stdio.h>   /* needed for printf() */

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

    printf("my process ID is %d\n", getpid());

    exit(0);
}
```

2. **system call `getpid()` to retrieve the PID of the current process.-**

```
/* GettingStarted/getpid.c */
#include <stdio.h>
#include <unistd.h>

int main(int argc,char *argv[])
{
    printf("My PID is %d\n",getpid());
}
```

What happens if you run the program multiple times ---Do PID change?

3. **fork, the original process** splits **into two, the original one, and the new created one.**

Here is an example

```
                /* LetsFork.c */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc,char *argv[])
{
 printf("Before Fork, My PID: [%d]\n",getpid());
 fork();
 printf("After Fork, My PID: [%d]\n",getpid());
```

```
 return 0;
}
```

What is the output of the above program ?

How do we distinguish between the child and parent process?

1. The child process will be spawned after `fork`.
2. The child will continue executing the code after `fork()` is returned, **not** from the beginning.
3. The parent **still** continues executing the same program.

Here is another practice case : Do it your self

```
/* LetsFork3.c */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc,char* argv[])
{
    int res;
    printf("Before fork():PID[%d]\n",getpid());
    res = fork();
    printf("The value of res is %d\n",res);
    if(res == 0)
    {
        printf("I am child! PID: [%d]\n",getpid());
    }
    else
    {
        printf("I am parent! PID: [%d]\n",getpid());
    }
    printf("Program Terminated!\n");
    return 0;
}
```

In the above case: The original process has the PID XXXX. After `res = fork();`, the process splits into two, the **parent** and **child**.

For the **parent**, the return value of `fork()` is the *PID of the new **child***. However, for the **child**, the return value is ***0***.
By using a condition, we can distinguish **parent** and **child**

4. **This program prints its process ID number and its parent's prcess ID number and then exits.**

```
#include <stdlib.h>      /* needed to define exit() */
#include <unistd.h>      /* needed to define getpid() */
#include <stdio.h>       /* needed for printf() */

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

        printf("my process ID is %d\n", getpid());
        printf("my parent's process ID is %d\n", getppid());
        exit(0);
}
```

5. https://www.digitalocean.com/community/tutorials/execvp-function-c-plus-plus
6. https://www.oreilly.com/library/view/secure-programming-cookbook/0596003943/ch01s07.html

More watching [this will help more and not boring like notes]

https://www.youtube.com/watch?v=Mqb2dVRe0uo

https://www.youtube.com/watch?v=6u_iPGVkfZ4
https://www.youtube.com/watch?v=OVFEWSP7n8c

https://www.youtube.com/watch?v=83M5-NPDeWs
https://www.youtube.com/watch?v=IFEFVXvjiHY
https://www.youtube.com/watch?v=6u_iPGVkfZ4&t=112s

https://www.youtube.com/watch?v=tcYo6hipaSA
https://www.youtube.com/watch?v=cex9XrZCU14&list=PLfqABt5AS4FkW5mOn2Tn9ZZLLDwA3kZUY

https://www.ibm.com/docs/en/aix/7.1?topic=e-exec-execl-execle-execlp-execv-execve-execvp-exect-fexecve-subroutine

https://www.youtube.com/watch?v=iq7puCxsgHQ