Experiment No. 8

Problem Definition:

Suppose two processes, parent and child, try to access a shared resource such as stdout. The output may not be desirable, called as race condition which occurs due to the order in which the processes are scheduled internally. Develop a C/C++ program to illustrate the race condition.

CSE@GIT

Objectives of the Experiment:

1) To familiarize with process creation

2) To understand how the resources are allocated to a process

3) To understand race condition

CSE@GIT

Need of the Experiment

To illustrate race condition.

Theoretical Background of the Experiment

Parent process

First, every process has a parent process (the initial kernel-level process is usually its own parent).

The parent is notified when the child terminates, and the parent can obtain the child's exit status.

Fork function

- An existing process can create a new one by calling the fork function.

```
#include <unistd.h>
pid_t fork(void);
```

Returns: 0 in child, process ID of child in parent.

- The new process created by fork is called the child process. This function is called once but returns twice. The only difference in the returns is that the return value in the child is 0, whereas the return value in the parent is the process ID of the new child.

RACE Condition

- Race condition occurs when multiple processes are trying to do something with shared data and the final outcome depends on the order in which the processes run.
- The fork function is a source for it. Depends on whether the parent or child runs first after the fork.
- In general, we cannot predict which process runs first. Even if we knew which process would run first, what happens after that process starts running depends on the system load and the kernel's scheduling algorithm.

Buffering

A **buffer** is a reserved area of the main memory which holds one **block**. When two or more **blocks** need to be transferred from disk to main memory.

Set the standard output unbuffered, so every character output generates a write.

Flow of implementation :

1. Declare required header files unistd.h, stdio.h.
2. Create the child process & check the condition for fork error.
3. If the condition is successful calls the charAtatime function.
4. Set the standard output unbuffered, so every character output generates a write.
5. It writes character by character onto the output & displays it on screen.

Pseudo Code / Outline of the Algorithm

```
pid=fork();

if (pid < 0)
{

  printf("fork error");

}
```

Pseudo Code / Outline of the Algorithm

```
if (pid == 0)
   {
        charatatime("cccccccccccccccccccc\n");
   }
else
  {
        charatatime("pppppppppppppppppppp\n");
   }
```

Pseudo Code / Outline of the Algorithm

```
static void charatatime(char *str)
{
 char *ptr;
 int c;
 setbuf(stdout, NULL); /* set unbuffered */
 for (ptr = str; (c = *ptr++) != 0; )
   putc(c, stdout);
}
```

RACE CONDITION

- The program outputs two strings: one from the child and one from the parent.
- The program contains a race condition because the output depends on the order in which the processes are run by the kernel and for how long each process runs.
- The goal in this example is to allow the kernel to switch between the two processes as often as possible to demonstrate the race condition.

OUTPUT

```
$ ./a.out
   ppppppppcccccccccccccccccccc
   pppppppppppp

$ ./a.out
   pppcpcppcppcppcppcppcppcppcppc
   cccccccccc

$ ./a.out
   pcccccccccccccccccccc
   ppppppppppppppppppp
```

Learning Outcomes of the Experiment

At the end of the session, students should be able to :

1) Understand context switch between processes [L2].
2) Understand sharing of resource between processes [L2].