CS 332 - Operating Systems
Fall 2012

October 3, 2012

Race Conditions

The learning objective of this task is to gain the first-hand experience on the race conditions. A *race condition* occurs when multiple processes access and manipulate the same data concurrently, and the outcome of the execution depends on the particular order in which the access takes place.

As a simple example let us assume that two processes P and Q, each want to output a string to `stdout`.

Program 1

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

static void charatatime(char *);

int main(void)
{
  int pid;

  if( (pid = fork()) < 0)
  {
    perror("Fork error");
    exit(1);
  }
  else if( pid == 0)
  {
    charatatime("Output from Process Q\n");
  }
  else
  {
    charatatime("Output from Process P\n");
  }
  exit(0);
}

static void charatatime(char * str)
{
  char * ptr;
  int c;
  setbuf(stdout,NULL);
  for(ptr = str; c = *ptr++;)
    putc(c, stdout);
```

```
}
```

The above 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 desired outcome will either have process P output its string followed by Q or vice versa. However, the output strings in `stdout` may be overlapped.

Race conditions when using fork and file descriptors

When forking a child process, file descriptors are copied to the child process, which can result in concurrent operations on the file. Concurrent operations on the same file can cause data to be read or written in a nondeterministic order, creating race conditions and unpredictable behavior.

As a example consider the following code which opens a file, read a character, fork, and then have both parent and child process read the second character of the file independently. However, because both processes share a file descriptor, one process might get the second character, and one might get the third. Furthermore, there is no guarantee the reads are atomic and the processes might get unpredictable results.

Program 2

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

void main()
{
  char c;
  int pid;

  int fd = open("input.txt", O_RDONLY);

  read(fd, &c, 1);
  printf("root process:%c\n",c);

  pid = fork();
  if (pid == 0)
  {
    read(fd, &c, 1);
    printf("child:%c\n",c);
  }
  else
  {
    read(fd, &c, 1);
    printf("parent:%c\n",c);
  }
}
```

Hints to fix the problem

One way to avoid the race condition in the above program is as follows:

- Closes the file descriptor in the child after forking
- Reopen the file in child, creating new file descriptor
- Perform read operation in child with new file descriptor
- Close the file descriptor after child finishes reading

Remarks

- The key to avoid race conditions is to prohibit more than one process from reading or/and writing the shared data at the same time.
- Depending on what file is being read and how important the order of read operations is, this problem can be dangerous.

Task

- In this lab, use the given hints to fix the race condition in program 2
- A more comprehensive problem on race conditions will be emailed by end of this week with detailed instructions.

***