Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

Communication between processes - files

Goal: We are going to understand some important features of child processes through examples. Firstly, we are not able to use simple local variables to pass data from one process to the other. Secondly, we show that the order of execution is not foreseen by us (it depends on the scheduler) and this fact causes some problem in making communication through common files. We show how to use advisory lock to avoid this problem. We will examine a strange problem with using random numbers in several processes.

We learn/repeat: *fcntl– creating an advisory lock (include unistd.h, fcntl.h); F_SETLK, F_SETLKW – constants for lock; F_RDLCK, F_WRLCK, F_UNLCK – locking for read, write or unlock; srand, rand – creates random numbers (include time.h)*

Tasks

1. Write a C program which has got an integer variable with a starting value before fork call! Change its value in the child process and check whether you can see the modified value in parent process or not! (*Remember, a child process has got an own working area with only copied values! So after fork there are two independent representatives of the original single variable and they are not able to read each others value!*)

2. Write a C program which has got a child process! Both the parent and the child process write texts (text1, text2) 25 times to the same file! Please, write the text character by character to see the effect – like in the part-code undergiven. (*Let you see the result of it! The file content will contain the texts quite mixed! The order of writing depends on the scheduler!)*

3. Modify the abovewritten program to avoid mixed texts by using advisory lock – you should force each process to finish writing out the text at least once! (*You have to use the predefined flock structure to make a lock by giving actual values to the fields – after it you have to call fcntl function to lock the file for a given process as you can see below.)*

```
int fcntl(int fd, int cmd, ... /* arg */ );
//fd – file descriptor
// command – F_SETLK – set the value given in an flock struct
// F_SETLKW – set the value and wait for execution,
// F_GETLK – get tehe vale
struct flock {
            ...
            short l_type;    /* Type of lock: F_RDLCK,
                                F_WRLCK, F_UNLCK */
            short l_whence;  /* How to interpret l_start:
                                SEEK_SET, SEEK_CUR, SEEK_END */
            off_t l_start;   /* Starting offset for lock */
            off_t l_len;     /* Number of bytes to lock */
            pid_t l_pid;     /* PID of process blocking our lock
                                (set by F_GETLK and F_OFD_GETLK) */
             };
```

4. Write a C program and a child process in it! Both the parent and the child should create 50 random numbers! *(If you use srand(time(NULL)) – as it is used usually, then almost sure you will get the same random numbers! The fact is that srand gets the time – and only secs for getting a random seed number. Starting a child process costs a very small amount of time, that is why you will get the same random numbers! We advise you not to use time value but e.g. the PID numbers, because they are different!)*

```
void srand(unsigned int seed)
//seed – a starting number for generating a random number list
int rand(void); //next random number
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government