Chapter 59: Interprocess Communication (IPC)

Inter-process communication (IPC) mechanisms allow different independent processes to communicate with each other. Standard C does not provide any IPC mechanisms. Therefore, all such mechanisms are defined by the host operating system. POSIX defines an extensive set of IPC mechanisms; Windows defines another set; and other systems define their own variants.

Section 59.1: Semaphores

Semaphores are used to synchronize operations between two or more processes. POSIX defines two different sets of semaphore functions:

1. 'System V IPC' — `semctl()`, `semop()`, `semget()`.
2. 'POSIX Semaphores' — `sem_close()`, `sem_destroy()`, `sem_getvalue()`, `sem_init()`, `sem_open()`, `sem_post()`, `sem_trywait()`, `sem_unlink()`.

This section describes the System V IPC semaphores, so called because they originated with Unix System V.

First, you'll need to include the required headers. Old versions of POSIX required `#include <sys/types.h>`; modern POSIX and most systems do not require it.

```
#include <sys/sem.h>
```

Then, you'll need to define a key in both the parent as well as the child.

```
#define KEY 0x1111
```

This key needs to be the same in both programs or they will not refer to the same IPC structure. There are ways to generate an agreed key without hard-coding its value.

Next, depending on your compiler, you may or may not need to do this step: declare a union for the purpose of semaphore operations.

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};
```

Next, define your *try* (`semwait`) and *raise* (`semsignal`) structures. The names P and V originate from Dutch

```
struct sembuf p = { 0, -1, SEM_UNDO}; # semwait
struct sembuf v = { 0, +1, SEM_UNDO}; # semsignal
```

Now, start by getting the id for your IPC semaphore.

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
    /* error handling code */
}
```

In the parent, initialise the semaphore to have a counter of 1.

```
union semun u;
u.val = 1;
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the value of
the semaphore to that specified by the union u
    /* error handling code */
}
```

Now, you can decrement or increment the semaphore as you need. At the start of your critical section, you decrement the counter using the `semop()` function:

```
if (semop(id, &p, 1) < 0) {
    /* error handling code */
}
```

To increment the semaphore, you use `&v` instead of `&p`:

```
if (semop(id, &v, 1) < 0) {
    /* error handling code */
}
```

Note that every function returns `0` on success and `-1` on failure. Not checking these return statuses can cause devastating problems.

Example 1.1: Racing with Threads

The below program will have a process `fork` a child and both parent and child attempt to print characters onto the terminal without any synchronization.

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

int main()
{
    int pid;
    pid =  fork();
    srand(pid);
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        char *s = "abcdefgh";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
        }
    }
```

```
    else
    {
        char *s = "ABCDEFGH";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
        }
    }
}
```

Output (1st run):

```
aAABaBCbCbDDcEEcddeFFGGHHeffgghh
```

(2nd run):

```
aabbccAABddBCeeCffgDDghEEhFFGGHH
```

Compiling and running this program should give you a different output each time .

Example 1.2: Avoid Racing with Semaphores

Modifying *Example 1.1* to use semaphores, we have:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define KEY 0x1111

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(KEY, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
```

```
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    srand(pid);
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        char *s = "abcdefgh";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            }
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }

            sleep(rand() % 2);
        }
    }
    else
    {
        char *s = "ABCDEFGH";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(15);
            }
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }

            sleep(rand() % 2);
        }
    }
}
```

Output:

```
aabbAABBCCccddeeDDffEEFFGGHHgghh
```

Compiling and running this program will give you the same output each time.