Exercise sheet 6 – Process synchronisation 1

Goals:

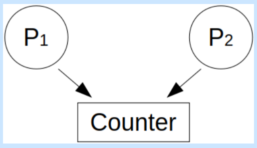
- Understand synchronisation issues
- Use semaphore for mutual exclusion
- Use Lock-Files for mutual exclusion

Exercise 6.1: Synchronisation problem analysis: theoretical

Consider two processes that counts information. Each process works independently. There is a `counter` file that hold the current state of the `counter`. Every time a process counts something, it opens the `counter` file, reads the current value, increases the value by one, and finally writes the new `counter` value into the `counter` file.

(a) Create a drawing that illustrates the situation (as simple as possible).

Proposal for solution:



(b) Write pseudocode to further illustrate the work of each process (as simple as possible).

Proposal for solution: Pseudocode for a process:

```
process() {
    while(1) {
        int counter = readfile("counter");
        ++counter;
        writefile("counter", counter);
    }
}
```

(c) What will happen if both processes work as described?

Proposal for solution: Both processes will most likely interfere with each other and will overwrite the change that the other process made.

(d) How could you solve the issue? Extend your pseudocode to solve the issue. Hint: you may use `P()/V()` operations.

Proposal for solution: Pseudocode for a process:
A semaphore `s` has to be created before the processes start.

```
seminit(s, 1);
process() {
    while(1) {
        P(s);
        int counter = readfile("counter");
        ++counter;
        writefile("counter", counter);
        V(s);
    }
}
```

Exercise 6.2: Synchronisation problem analysis: practical

(a) Update the `OS_exercises` repository with `git pull`.

Proposal for solution: `git pull`

(b) Change into the `OS_exercises/sheet_06_process_sync1/counting_sem` directory.

Proposal for solution: `cd OS_exercises/sheet_06_process_sync1/counting_sem`

(c) Inspect the `counting_process.c` file.

(d) If you would start two processes of the `counting_process` and the initial value in the `counter` file was 0, which value should be in the `counter` file after both processes ended?

Proposal for solution: The counter should be at 200000.

(e) Print the value of the `counter` file on the shell.

Proposal for solution: `cat counter`
The counter is at 0.

(f) Start two processes of `counting_process` in parallel on the shell (if it takes too long: reduce the number inside the for loop and compile again). Use the provided `start.sh` for that by calling `./start.sh`. The `start.sh`

- Resets the `counter` file to 0
- Starts two processes of `counting_process`
- Waits until both processes have finished
- Prints the value of the `counter` file

Proposal for solution:

```
./start.sh
```

(g) What is the value of the `counter` file? Have you expected that?

Proposal for solution: `cat counter`
The counter is below the expected 200000. There may be some synchronisation issues.

Exercise 6.3: Synchronisation with a semaphore

(a) Make sure you are in the `OS_exercises/sheet_06_process_sync1/counting_sem` directory.

(b) Compile your `counting_process.c` into `counting_process`, just to make sure everything compiles. Use the provided `Makefile` for that.

Proposal for solution: `make`

(c) Use a semaphore to fix the synchronisation issue in `counting_process.c`.

Proposal for solution:

```c
#include <stdio.h>     //printf, perror
#include <stdlib.h>    //EXIT_FAILURE, EXIT_SUCCESS
#include <string.h>    //sprintf
#include <unistd.h>    //open, close, read, write
#include <fcntl.h>     //flags: O_CREAT, O_EXCL
#include <semaphore.h> //sem_open, sem_wait, sem_post, sem_close, sem_unlink
#include <errno.h>     //errno

#define SEMAPHORE_NAME "/global_counter" //Name of semaphore
sem_t*   semaphore = NULL;               //Pointer to semaphore
const int     PERM = 0600;               //Permission to the semaphore (read + write)

void create_semaphore() {
    semaphore = sem_open(SEMAPHORE_NAME, O_CREAT, PERM, 1);
    if(semaphore == SEM_FAILED) {
        perror("Error when creating the semaphore ...\n");
        exit(EXIT_FAILURE);
    }
}

void delete_semaphore() {
    if(sem_close(semaphore) == -1) {
        perror("Error can't close semaphore ...\n");
        exit(EXIT_FAILURE);
    }

    if(sem_unlink(SEMAPHORE_NAME) == -1) {
        switch(errno)
        {
        case EACCES:       //Fall through
        case ENAMETOOLONG:
            perror("Error can't delete (unlink) semaphore ...\n");
            exit(EXIT_FAILURE);
            break;
        case ENOENT: //Semaphore already deleted, no error should be printed!
            break;
        }
    }
}

int main () {
    create_semaphore();

    //Main task: Loop 100000 times and add 1 to the counter inside the loop
    for (int i = 0; i < 100000; ++i){
        //Lock the semaphore
        sem_wait(semaphore); //P(s)

            //Open the file
            int file = open("counter", O_RDWR);
            if (file == -1) {
                printf("Could not open file, exiting!\n");
                exit(EXIT_FAILURE);
            }

            //Read the number
            const int MAX_LEN = 64;
```

```c
                char number[MAX_LEN];
                read(file, &number, sizeof(number));

                //Convert the string into an integer
                int counter = atoi(number);
                counter++;

                //Write the new number into the counter
                sprintf(number, "%d\n", counter);
                lseek(file, 0, 0);
                write(file, &number, strlen(number) + 1);

                //Close the file
                close (file);

            //Unlock the semaphore
            sem_post(semaphore); //V(s)
        }

        delete_semaphore();

        printf("Finished!\n");

        return EXIT_SUCCESS;
}
```

(d) Compile your `counting_process.c` again.

Proposal for solution: `make`

(e) Start two processes of `counting_process` in parallel on the shell (if it takes too long: reduce the number inside the for loop and compile again). Use the provided `start.sh` for that by calling `./start.sh`.

Proposal for solution:

```
./start.sh
```

(f) What is the value of the `counter` file? Have you expected that?

Proposal for solution: `cat counter`
Now, the counter should have the expected 200000 and the synchronisation issue should be fixed.

Exercise 6.4: Synchronisation with a lock file (optional)

(a) Change into the `OS_exercises/sheet_06_process_sync1/counting_flock` directory.

Proposal for solution: `cd OS_exercises/sheet_06_process_sync1/counting_flock`

(b) Compile your `counting_process.c` into `counting_process`, just to make sure everything compiles. Use the provided `Makefile` for that.

Proposal for solution: `make`

(c) Use a `counter.lck` lock file and the `flock()` function to fix the synchronisation issue in `counting_process.c`

Proposal for solution:

```c
#include <stdio.h>     //printf, perror
#include <stdlib.h>    //EXIT_FAILURE, EXIT_SUCCESS
#include <string.h>    //sprintf
#include <unistd.h>    //open, close, read, write
#include <fcntl.h>     //flags: O_CREAT, O_EXCL
#include <errno.h>     //errno
#include <sys/file.h>  //flock
#include <sys/stat.h>  //umask

const int     PERM = 0600;  //Permission to lock file: counter.lck (read + write)

int main () {
    //Main task: Loop 100000 times and add 1 to the counter inside the loop
    //Open and create the file counter.lck
    umask(0177); //Makes sure, new files can only be read/write by the user
    int file_lock = open("counter.lck", O_WRONLY | O_CREAT, PERM);

    for (int i = 0; i < 100000; ++i) {
        flock(file_lock, LOCK_EX); //P(s)

            //Open the file
            int file = open("counter", O_RDWR);
            if (file == -1) {
                printf("Could not open file, exiting!\n");
                exit(1);
            }

            //Read the number
            const int MAX_LEN = 64;
            char number[MAX_LEN];
            read(file, &number, sizeof(number));

            //Convert the string into an integer
            int counter = atoi(number);
            counter++;

            //Write the new number into the counter
            sprintf(number, "%d\n", counter);
            lseek(file, 0, 0);
            write(file, &number, strlen(number) + 1);

            //Close the file
            close(file);

        flock(file_lock, LOCK_UN); //V(s)
    }

    close(file_lock);

    printf("Finished!\n");

    return EXIT_SUCCESS;
}
```

(d) Compile your `counting_process.c` again.

Proposal for solution: `make`

(e) Start two processes of `counting_process` in parallel on the shell (if it takes too long: reduce the number inside the for loop and compile again). Use the provided `start.sh` for that by calling `./start.sh`.

(f) What is the value of the `counter` file? Have you expected that?

Proposal for solution: `cat counter`
Now, the counter should have the expected 200000 and the synchronisation issue should be fixed.