DM510 Operating Systems
Spring 2026

Exercise Sheet 2

Complete before tutorial on Thursday, February 19th

Learning goals

Be able to

- explain programs, processes, process ids
- explain context switches
- understand the functioning of the `fork()` system call
- describe inter-process communication via shared memory and message passing and their implementations in C (Linux)
- describe the high level functioning of static and shared libraries and their usage in C (Linux)

Chapter 3

Exercise 1. Describe the actions taken by a kernel to context-switch between processes.

Exercise 2. Some computer systems provide multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

Exercise 3. When a process creates a new process using the `fork()` operation, which of the following states is shared between the parent process and the child process?

- Stack
- Heap
- Register values
- Shared memory segments, as for example created in the lecture with `mmap()` and flag `MAP_SHARED`

Exercise 4. Including the initial parent process, how many processes are created by the program shown below?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        /* fork a child process */
        fork();
        /* fork another child process */
        fork();
        /* and fork another */
        fork();
        return 0;
}
```

Exercise 5. Using the program below, identify the values of pid at lines `A`, `B`, `C`, and `D`. Assume that the actual pids of the parent and child are 2600 and 2603, respectively.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
        pid_t pid, pid1;
        /* fork a child process */
        pid = fork();
        if (pid == 0) {
                /* child process */
                pid1 = getpid();
                printf("child: pid = %d\n",pid);    /* A */
                printf("child: pid1 = %d\n",pid1); /* B */
        }
        else {
                /* parent process */
                pid1 = getpid();
                printf("parent: pid = %d\n",pid);    /* C */
                printf("parent: pid1 = %d\n",pid1); /* D */
                wait(NULL);
        }
        return 0;
}
```

Exercise 6. Named pipes (created with the system program `mkfifo`) and ordinary pipes (created with the system call `pipe()`) are two ways of implementing simple message passing in Linux. Give an example where each is more suitable than the other.

Exercise 7. What is synchronous and asynchronous communication? Name advantages and disadvantages of each.

Exercise 8. Compare advantages and disadvantages of shared memory and message passing as means of inter-process communication.

C Programming

The following exercises assume that you have access to a Linux system. You can for example use your Raspberry Pi.
Create the following directory structure:

```
exer2/main.c
exer2/libpow/shift.c
exer2/libpow/rec.c
exer2/libpow/include/pow.h
```

`shift.c` should contain:

```
#include "include/pow.h"

int power_shift(int x, unsigned int exp)
{
     float result;
     /* base case */
     if (exp == 0)
         return 1;

     result = power_shift(x, exp >> 1);
     result = result * result;

     if (exp & 1)
         result = result * x;
     return result;
}
```

`rec.c` should contain:

```
#include "include/pow.h"

int power_rec(int x, unsigned int exp)
{
     /* base case */
     if (exp == 0) {
         return 1;
     }
     /* recursive case */
     return x*power_rec(x, exp - 1);
}
```

`pow.h` should contain:

```
#ifndef __POW_H__
#define __POW_H__

int power_shift(int x, unsigned int exp);

int power_rec(int x, unsigned int exp);

#endif
```

Exercise 9. Discuss the two implementations of a raising a number `x` to the power of `exp`. Which do you believe is more readable and which is more efficient and why?

Exercise 10. Create a makefile `exer2/libpow/Makefile` that builds a static library `libpow.a` that contains the two power implementations. To build a static library, you first need to compile the source files to object files. Then a static library can be built using `ar rcs <libname>.a <object1>.o <object2>.o ....`

Exercise 11. Create a file `exer2/main.c` that starts with `#include <pow.h>` and that has a main function, which tests the two functions. When compiling it, you need to tell the compiler where to find the header file of the library. To do so, pass the parameter `-Ipath/to/header/directory` to `gcc`.

Exercise 12. Change the makefile `exer2/libpow/Makefile` so that it also build a dynamic/shared libary. To build a shared library, you again first need to compile the source files to object files. Then a shared library can be built using `gcc -shared -o <libname>.so <object1>.o <object2>.o ....`

Exercise 13. In `exer2/Makefile` create a new target `main-shared`, where you dynamically link `main.o` to `libpow.so`. To do so, you need to invoke `gcc` and pass `-l<name>` for the libary. Further, you need to tell `gcc` where to find shared libaries by passing `-Lpath/to/libraries`. The full command is `gcc -Llibpow -o main-shared main.o -lpow`. Execute `main-shared` by running `LD_LIBARY_PATH=libpow ./main-shared`. The first part is to tell the loader where to find the shared libraries.

Exercise 14. Discuss

a. What are libraries needed for?

b. What is the difference between static and shared (dynamically linked) libraries?

c. Name an advantage of static libraries and one of shared libraries.