Operating Systems 2016/17
Tutorial-Assignment 2

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Question 2.1: Multi-Programming

a. Explain the difference between single-programming and multi-programming systems. What is the advantage of multi-programming?

b. Explain the difference between CPU-bound and I/O-bound processes.

c. Why is a good mixture of CPU-bound and I/O-bound processes preferable?

Question 2.2: Processes in Unix

a. What keeps a process from accessing the memory contents of another process?

b. What are typical regions in a process address space? What is their purpose?

c. What does the `fork()` system call do?

d. Write a small C program that creates a child process. Each process shall print out who it is (i.e., parent or child). The parent shall also print out the child's PID and then wait for the termination of its child.

e. Assume you have to write a shell that can be used to launch arbitrary other programs. Is the `fork()` system call sufficient for that purpose?

Question 2.3: Stacks and Procedures in C

a. Discuss the following code fragment. Try to visualize the stack contents before, during, and after the execution of `foo`. All values are passed via the stack between calling and called function. An `int` is 4 bytes and a `double` is 8 bytes long. Assume a 4-byte aligned, downwards growing pre-decrement stack and the existence of a stack-frame pointer. All local entities within a function are addressed relative to this frame pointer.

```
double foo ( int *p )
{
    int x;
    double y;
    x = *p;
    // do something useful
    return y;
}
```

```
double bar ()
{
   double d;
   int i = 42;
   d = foo( &i );
   return d;
}
```

b. Characterize as briefly and precisely as possible the difference between a function and a macro in C. Outline the consequences of these two different implementations with regard to the caller.