Prof. Dr. Burkhard Stiller, University of Zurich, Binzmühlestrasse 14, CH-8050 Zürich
Telephone: +41 44 635 6710, Fax: +41 44 635 6809, stiller@ifi.uzh.ch
Bruno Rodrigues, Telephone: +41 44 635 4381, rodrigues@ifi.uzh.ch
Sina Rafati, Telephone: +41 44 635 6702, rafati@ifi.uzh.ch

Exercises in SSDS

Exercise 3

Hand out date: Mar 04, 2020
Submission until and discussion: Mar 12, 2020 08:15

Processes, Signals and Pipes

1) Theoretical Questions

1.1) Tick the **correct** alternative. In a typical Operating System (OS), the states of a process are:

	Active, Ready, Unnocupied, Terminated
	Locked, Unlocked, Active and Suspended
	Running, Locked and Ready
	New, Ready, Running, Waiting, Terminated
	Ready, Terminated, Active, Processing

1.2) Check the **correct** alternative that represents a transition between processes states that is **not** allowed.

	Ready for Running
	Running for Waiting
	Running for Terminated
	New for Ready
	Ready for Terminated

1.3) Consider the execution of three processes in the following situation:

- Process P1 sends a message to process P2.
- Process P2, upon receiving the message from P1, responds to that message to P1.
- Process P1, upon receiving the reply message from P2, responds to P2 with a new message, and so on.
- Process P3 is blocked, a situation of which will only occur when receiving a message from process P1.

Considering that the priority of process P3 is lower than the priorities of processes P1 and P2, mark the **correct** alternative:

	In each exchange of messages between P1 and P2 the respective priorities will ber automatically reduced, and when they are lower than that of process P3, it will be executed
	After 1 second locked, the real-time clock of the operating system will automatically give the P3 process the opportunity to be unlocked.
	Process P3 will exit this situation as soon as any interruption occurs
	This is a situation known as *deadlock*
	This is a situation known as *starvation*

1.4) Check the **correct** alternative that corresponds to a valid state for a process in an operating system.

	Waiting, for example, the completion of an I/O task
	Blocked, when the processor starts the process execution for the first time
	Executing, when the compilation of its source code by the scheduler begins
	Zombie, when the process is waiting for the execution of itself by all processors, before self-locking by preemption
	Enrolled, when a new process is being created

1.5) The state describing a process “asleep”, on hold due to the occurrence of an external event, characterizing the voluntary delivery of the processor use to the operating system is called. Mark the **correct** alternative:

	Blocked
	Waiting
	Dispatch
	Running
	Ready

1.6) About the different processes states in an operating system, mark the **incorrect** alternative:

	A process, after being admitted, remains suspended waiting for a resource that is not available or an I/O data
	Some examples of process states are: suspended, ready, running, and waiting
	The process, when terminated, means that it has finished running and can be removed from system memory
	A process, after the waiting state, can return to the running state after it is ready
	A process is new when it is being created, that is, its code is being loaded into memory, along with its libraries

1.7) In the Inter-Process Communication of an operating system, there are some possible problems, such as race conditions. What is this problem about? Tick the **correct** alternative.

	Measurement of speed of execution of processes
	Amount of memory used by each process
	Performance of each process in relation to its previous one
	Processes wanting to access a shared memory area at the same time
	Memory access speed of each process involved

1.8) In modern operating systems, communication between processes is allowed, which can be manifested through various mechanisms. One of these mechanisms is called (mark the **correct** alternative):

	Direct Memory Access (DMA)
	Interruptions
	Shared Memory
	Virtual Memory
	Subroutines

1.9) In the process management of most UNIX systems, mark the **correct** alternative which corresponds to the system call named *fork*:

	Sends a signal to a process
	Defines the action to be taken upon receiving a particular signal
	Terminates the current process
	Creates a child process identical to the parent process
	Waits until the end of the child process

1.10) Regarding Inter-Process Communication (IPC) of an operating system. If a process P1 wants to send data to process P2, process P1 writes on one side (e.g., just as it was writing to a file) and process P2 can read the data as if it were reading from an input file. This IPC method is called (mark the **correct** alternative):

	Channel
	Pipe
	Queue
	Thread
	Stdout

1.11) Concerning the available IPC mechanisms, such as Pipes, Sockets, Shared Memory and Remote Procedure Calls (RPC). These are implemented by an OS as the following subsystem:

	File System Manager
	Manager of Inter-Process Communication
	Process Scheduler
	Memory Manager
	I/O Manager

1.12) Concerning the situations in which a race conditions is created, check the **correct** alternative:

	When processes run on machines whose processing speed differences are greater than cos (45 °) (approximately 70%).
	When 3 (three) or more processes are involved in accessing shared variables.
	Whenever Semaphores are used to protect the critical sections of the processes.
	When processes manipulate shared variables concurrently.
	Only when processes are deadlocked, caused by the protection mechanism of critical sections.

2) Code Analysis - Fork()

2.1) The fork() system call produces a new process each time is called. The new process created by fork() is a copy of the current process except for the returned value. Given the code below, calculate the number of times "Hello SDDS" is printed and ii) draw the tree of child processes created by the fork(), explaining the output.

```
#include  <stdio.h>
#include  <sys/types.h>
int main()
{
    fork(); //creates a child
    fork(); //parent and child will fork
    fork(); //similarly, the previous processes will fork
    printf("Hello SDDS!\n"); //all of them will execute this line
    return 0;
}
```

2.2) Consider a process executing the following code:

```
for(i = 0; i < n; i++)
    fork();
```

The total number of child processes given by:

	n
	2^n – 1
	2^n

	2^(n+1) – 1;
	2^(n+1) + 1;

2.3) Analyze the code below and describe the output. What is objective of the wait() method?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void _fork()
{
       int ret;
       ret = fork();

       if (ret == 0)
       {
           printf("\n [%d] Running Child \n", getpid());
           sleep(1);
           printf("\n [%d] Ending Child \n", getpid());
       }
       else
       {
           printf("\n [%d] Waiting Parent \n", getpid());
           wait(NULL);
           printf("\n [%d] Ending Parent \n", getpid());
       }
 }
int main ()
{
      _fork();
      return 0;
}
```

3) Signaling and Pipes

Signals is a notification sent by either operating system or some application to a running program. Signals are a limited form of inter-process communication used in Unix, Unix-like, and other POSIX-compliant operating systems. When a signal is sent, the operating system interrupts the target process' normal flow of execution to deliver the signal. Execution can be interrupted during any non-atomic instruction. If the process has previously registered a signal handler, that routine is executed. Otherwise, the default signal handler is executed. To see the list of available signals on Linux, type in the terminal:
$kill -l

a) Describe the function of the SIGINT, SIGKILL and SIGSTOP messages.

b) Describe the output of the code below when you execute it and interrupt the execution pressing CTRL+C

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

void sig_handler(int signum)
{
    printf("Received signal %d\n", signum);
}
```

```
int main()
{
    signal(SIGINT, sig_handler);
    sleep(10); // This is your chance to press CTRL-C
    return 0;
}
```

c) A Pipe is a mechanism for interprocess communication. Data written to the Pipe by one process can be read by another process. Create a Pipe to perform the interprocess communication between two processes using the methods pipe(), write() and read(). Write in the pipe the following messages:

int fd[1]; //declaring a Pipe file descriptor
msg1 = “Hello world #1”;
msg2 = “Hello world #2”;
msg3 = “Hello world #3”;

d) Create a pipe to exchange a string message to a child process. The first step is to create the pipe() and then use the fork() to create a child process. The father process should write() in the pipe and then the child process should read().

Submission Guidelines

- Submission is not mandatory.
- Zip your code using the name pattern: ssds_ex+**exercise+number_initials**. For instance, the file of Exercise 01 for Bruno Rodrigues would be named: sssds_ex01_BR.zip
- Submit to rodrigues@ifi.uzh.ch