Department of Informatics IfI
University of Zurich
Binzmühlestrasse 14
CH—8050 Zürich
Switzerland
URL: https://www.ifi.uzh.ch

Prof. Dr. Burkhard Stiller
Bruno Rodrigues
Rafael Ribeiro
Communication Systems Group
CSG
Phone: +41 44 635 46 81
E-mail: [lastname]@ifi.uzh.ch
URL: https://www.csg.uzh.ch

3121- Systems Software and Distributed Systems (SSDS)

E3 - Processes and IPC

1 Theoretical Exercises

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

   [_] Active, Ready, Unoccupied, Terminated
   [_] Locked, Unlocked, Active and Suspended
   [_] Running, Locked and Ready
   [✓] New, Ready, Running, Waiting, Terminated
   [_] Ready, Terminated, Active, Processing

2. Check the correct alternative representing a transition between processes states that is **not** allowed.

   [_] Ready to Running
   [_] Running to Waiting
   [_] Running to Terminated
   [_] New to Ready
   [✓] Ready to Terminated

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 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 be 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.

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

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

6. Concerning 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

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

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

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

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

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

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 degrees) (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.

13. (Book's question): What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

    - Synchronous and asynchronous communication

      ____

      ____

      ____

- Automatic and explicit buffering

- Send by copy and send by reference

- Fixed-sized and variable-sized messages

14. (Book's question): Describe the actions taken by a kernel to context-switch between processes.

15. (Book's question): Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

16. (Book's question): Describe the differences among short-term, medium-term, and long-term scheduling.

17. (Book's question): Does a pipe allow unidirectional communication or bidirectional communication?

18. (Book's question): Can the pipes communicate over a network, or must the communicating processes reside on the same machine?

2 Practical Exercises

2.1 Code Analysis - Fork()

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. 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;

[__] $2^(n+1)+1$;

3. Analyze the code below and describe the output. What is the 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;
    }

```

2.2 Inter-Process Communication

Signals are a notification sent by either an 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's 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: $kill - l$

1. (Book's question): To which type of IPC communication the following code refers?

```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1
int main(void){
char write_msg[BUFFER_SIZE]
```

```
                char read_msg[BUFFER_SIZE];
                int fd[2];
                pid_t pid;
                ...
                ...
                }

```

[__] Sockets

[__] Shared Memory

[__] Write Receiver

[__] RPC

[__] Pipe

2. (Book's question): Similarly, to which type of IPC communication the code below refers?

```
        #include <stdio.h>
        #include <syslshm.h>
        #include <syslstat.h>
        int main()
        {
            int segment_id;
            char *sm;
            const int size = 4096;
            125
            segment_id = shmget(IPC_PRIVATE, size, s_IRUSR | s_IWUSR)
    ;
        }
        sm = (char*) shmat(segment_id, NULL, 0);
        sprint (sm, "Hi there!");
        * now print out the string  *
        printf(" *%s \n" , sm) ;
        * now detach the segment *
        shmdt(sm);
        * now remove the segment *
        shmctl(segment_id, IPC_RMID, NULL);
        return 0;

```

[__] Shared Memory

[__] Sockets

[__] Write Receiver

[__] RPC

[__] Named Pipe

3. Describe the function of the SIGINT, SIGKILL and SIGSTOP messages.

4. Describe the output of the code below when you execute it and interrupt the execution by 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;
        }

```

5. A Pipe is a mechanism for inter-process communication. Another process can read data written to the Pipe by one process. Create a Pipe to perform the inter-process 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";

```

6. 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().

3 Submission Guidelines

- Include ”SSDS” in the e-mail’s title.
- Any submitted code must compile without error.
- Zip your files using the name pattern: ssds_ex+exercise+number_initials (*e.g.,* ssds_ex01_BR.zip)
- Submit the exercise to the research assistant responsible for the task e-mail addresses.