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 5

Hand out date: Mar 18, 2020
Discussion: Mar 26, 2020 08:15

Critical Section, Semaphores and Deadlock

1) The Critical Section (CS) is a fundamental concept in Operating Systems. Concerning the access control of a CS, it is **correct** to state that (mark the correct answers):

	CS is a segment of code prune to race conditions in which shared variables can be accessed
	There is a mechanism to control access to the CS
	A process immediately enters the CS when it arrives
	A process outside the critical region may block a process within the critical region.
	There may be more than one process running within the critical region
	All options are false.

2) About the Deadlock concept, analyze the following assertions and **mark** the alternative points to the **correct** answer(s).

(I) The Deadlock problem exists in any multiprogrammable system. However, implemented solutions must consider the type of system and the impact on its performance. For example, a Deadlock in a real-time system, which controls a nuclear power plant, must be treated with mechanisms designed for this type of application, different from those adopted by a common time-sharing system.

(II) Deadlock improves the distribution of CPU time in relation to non-preemptive scheduling, but still can not implement an equitable sharing between the different types of processes.

(III) A process is in a Deadlock condition when it is waiting for an event that will never occur. This condition is a consequence, for the most part, of sharing system resources among various processes whereas each process must have exclusive access to the resource (mutual exclusion).

(IV) Deadlock can also be caused by a clock, which interrupts the processor at certain time intervals so that a CPU-bound process is more likely to run.

	Only (I)
	Only (I) and (III)
	Only (I), (III) and (IV)
	Only (II), (III) and (IV)
	All options

3) Regarding the conditions for a deadlock to occur, which resources previously granted to a process cannot be necessarily taken from that process, but rather explicitly released by it, it is called a condition of:

	Preemption
	Mutual exclusion
	Possession and waiting
	No preemption
	Circular waiting

4) Regarding the correction of a Deadlock situation, review the assertions and check the alternative that points to the **correct** (s).

(I) After the deadlock is detected, the system should correct the problem. A solution widely used by most operating systems is simply to eliminate one or more processes involved in the deadlock and deallocate resources already guaranteed by them, thus breaking the circular wait.

(II) A deadlock has no fix, since the running process waits for an event that will never occur again. The operating system must be able to identify and eliminate a deadlock process.

(III) A less drastic solution involves releasing only a few resources allocated to processes for other processes until the waiting cycle ends.

(IV) The elimination of the processes involved in the deadlock and, consequently the release of their resources may not be simple, depending on the type of resource involved.

	Only (I)
	Only (I), (II) and (III)
	Only (I), (III) and (IV)
	Only (II), (III) and (IV)
	All options

5) How many processes, at a minimum, may be involved in a deadlock of an operating system? Mark the **correct** alternative.

	1
	2
	3

	4
	5

6) Consider the following statements about synchronization and deadlock in multiprocessor and multiprogrammed computing system:

(I) Only the removal of the circular wait is sufficient to prevent the occurrence of deadlocks;

(II) Only the preemptive release of resources is enough to eliminate the possibility of deadlocks;

(III) The elimination of exclusive use of resources alone is sufficient to prevent the occurrence of deadlocks;

(IV) The possibility of deadlocks occurrence is eliminated only when there is no busy and circular wait

The number of **correct** statements is (mark the correct alternative):

	0
	1
	2
	3
	4

7) Regarding the situations in which a deadlock is created, the following conditions must occur simultaneously

	Mutual exclusion, hold and wait, not preemption and circular wait
	Mutual exclusion, thrashing, process overlapping and circular wait
	Excessive thrashing, process overlapping, hold and wait and no preemption
	Mutual exclusion, hold and wait, process overlapping and scheduling failures
	Excessive thrashing, no preemption, circular wait, and scheduling fail.

8) Regarding the preemptive and non-preemptive scheduling policies. Analize the following sentences.

(I) A scheduling algorithm is characterized as non-preemptive when, in the operating system, a running process is interrupted for another process to use the processor.
(II) In a preemptive policy, the scheduling occurs at the beginning of an interrupt handling, and depends on the willingness of the process to release the processor.
(III) Preemptive algorithms allows a process to be interrupted during its execution, either through an I/O, or due to the adopted scheduling policy and applied by the process scheduler or simply by force of the term implementation of the procedure.
(IV) Preemption is the act of temporarily interrupting a task being solved by a computer system, without the need of its cooperation, and with the intention of resuming the task afterwards. Such a change is also known as a Context Switch.

Review these assertions and check the alternative that points to the **correct** one(s):

	Only (I)

	Only (II)
	Only (I) and (II)
	Only (III)
	Only (III) and (IV)

9) The concept of semaphore was proposed by Dijkstra and shows, in a simple way, how to implement mechanisms between processes. Regarding this subject, tick the **correct** option.

	Binary semaphores (mutexes) may assume any positive integer value.
	The down instruction executed with value 0 causes the process to enter the standby state.
	A semaphore may assume negative values.
	Semaphore is a non-negative integer variable that, in some implementations can handle more than two values.
	Since they work with the OR logical connection, semaphores are unable to implement mutual exclusion due to the required change in this type of control.

10) Semaphores are used in concurrent programming to prevent two or more processes from accessing a Critical Section (CS) simultaneously. Thus, semaphores are used to ensure (**mark** the **correct** answers):

	Mutual exclusion, where each process should signal immediately before and, immediately thereafter, use a shared resource
	Pipes, which work by connecting the output of one process with the input of another
	The PID, the unique identification of each process
	That there will be no deadlock, *i.e.*, when one process waits indefinitely for a resource of another.
	That the virtual machine manages the processes of the Operating System and the hardware of the machine correctly

11) Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and - once finished - will return them. As an example, many commercial software packages provide a given number of licenses, indicating the number of applications that maz run concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be granted when an existing license holder terminates the application and a license is returned.

The following program segment is used to manage a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:

```
# define MAX_RESOURCES 5
int available_resources = MAX_RESOURCES
```

When a process wishes to obtain a number of resources, it invokes the *decrease_count()* function:

```
/* decrease available_resources by count resources */
/* return 0 if sufficient resources available, otherwise return -1 */
int decrease_count(int count){
     if(available_resources < count)
            return -1;
     else{
            available_resources -= count;
            return 0;
     }
}
```

When a process wants to return a number of resources, it calls the *increase_count()* function:

```
/* increase available resources by count */
int increase_count(int count){
     available_resources += count;
}
```

The preceding program segment produces a race condition.
a) Identify the data involved in the race condition.
b) Identify the location (or locations) in the code where the race condition occurs.
c) Using a semphore, fix the race condition. It is okay to modify the *decrease_count()* function so that the calling process is blocked until sufficient resources are available.

Submission Guidelines

- Submitted code must compile without errors.
- 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 the exercise to the reseach assistants e-mail addresses.