1 OPERATING SYSTEMS LABORATORY VI - Synchronization, Critical Section, InterProcessCommunications I

Examples:

- Compile and run the code.
- Analyze the code and output.
- You should compile with **-lpthread** whenever necessary.

1. **Signal**; code28.c

   - Signals are mechanisms for communicating with and manipulating processes.
   - A signal is a special message sent to a process. Signals are asynchronous; when a process receives a signal, it processes the signal immediately, without finishing the current function or even the current line of code.
   - Each signal type is specified by its signal number, but in programs, you usually refer to a signal by its name.
   - How to terminate the program? Break with Ctrl+Z, you will get

     ```
     [1]+ Stopped code28
     ```

     then kill the stopped process with

     ```
     kill %1
     ```

2. **Signal Handling**; - code29.c

   - Even assigning a value to a global variable can be dangerous because the assignment may actually be carried out in two or more machine instructions, and a second signal may occur between them, leaving the variable in a corrupted state.
   - If you use a global variable to flag a signal from a signal-handler function, it should be of the special type **sig_atomic_t**.
   - Assignments to variables of this type are performed in a single instruction and therefore cannot be interrupted midway.

- This program uses a signal-handler function to count the number of times that the program receives SIGUSR1, one of the signals reserved for application use.

3. **Semaphore**; code34.c

   - A common strategy to avoid race conditions is to use semaphores.
   - The use of semaphores is important to prevent simultaneous access to system resources by separate processes or separate threads inside the same process.
   - Three system calls to create, use, and release semaphores:
     - **semget** - Returns an integer semaphore index that is assigned by the kernel
     - **semop** - Performs operations on the semaphore set
     - **semctl** - Performs control operations on the semaphore set
   - The program shows how to create a semaphore set and how to access the elements of that set. Does the followings:
     - Creates a unique key and creates a semaphore
     - Checks to make sure that the semaphore is created OK
     - Prints out the value of the semaphore at index 0 (should be 1)
     - Sets the semaphore (decrements the value of semaphore at index 0 to 0)
     - Prints out the value of the semaphore at index 0 (should be 0)
     - Unsets the semaphore (increments the value of semaphore at index 0 back to 1)
     - Prints out the value of the semaphore at index 0 (should be 1)
     - Removes the semaphore
   - Study the code.
   - Execute several times and observe that how the output changes.
   - Is there any possible race conditions? Explain.

4. **Mutex**; code32.c

   - Several threads and shared data.

- Mutex mechanism (pthread_mutex_lock) is used for concurrent executing.
- Execute code several times and observe that how the execution order of the threads changes.