Göktuğ Akın
161044018

# CSE344
# SYSTEM PROGRAMMING

# HOMEWORK 2 REPORT

**Problem definition :**
In this project, we have two processes that are doing some calculations on 2D coordinates. These processes must run within mutual exclusion to finish the job gracefully.

**Solving the problem by dividing into sub-problems :**

**1) File protection**

Processes in this project, work on the same file to do calculations. Both processes are running concurrently, so there must be a protection mechanism that avoids conflicting between processes, for example, a process must waits to another process writing operation to finish. Otherwise, there will be a conflict.
By using fcntl library, we can achieve this protection mechanism. Both of these processes must lock the file before he operates on the file. After the completion of the job, he must unlock the file.

**2) Interprocess communication via signals**

In this project,  both processes use the signals for communicating between each other. Communication scenarios explained ;

- Process P1 send **SIGUSR1** to process P2 to indicate the process P2 that P1 is finished. P2 must know that P1 is alive or not since it will be exit by looking this condition. If temporary file is empty, P2 must check that P1 is finished or not.
- Process P1 send **SIGUSR2** to process P2 to indicate the process P2 that P1 writes some bytes on the common file (temporary file). By this way, If process P2 is suspended, when process P1 writes to temp file, P2 will be wake up and it can continue to do its job.

    - **Note** : **SIGUSR1** and **SIGUSR2** signals are initially blocked in the process P2. The reason is, there can be bad scenario. This bad scenario is, **SIGUSR1** or **SIGUSR2** can be caught in the process P2 while its NOT suspending. This means that, SIGUSR1 and SIGUSR2 (which are sent by P1) can be lost. If these signals are lost, P2 will never wake up. In order to avoid this, P2 blocks these signals except suspending. In case of sigsuspend, we use a mask that do not block **SIGUSR1** and **SIGUSR2**, so P2 will listen these signals just in the suspended case.
- Process P2 sends **SIGUSR1** and **SIGUSR2** signals to process P1 to indicate that P2 is started. By this way, P1 can know that P2 is started. This is system is important. Process P1 can send signal **SIGUSR1** or **SIGUSR2** to its child (P2) before P2 starts. But this is abnormal, this should not be happened. To avoid this, P2 send “I am started” signal to its parent (P1).
    - **SIGUSR1** : sent by P2 to P1. P1 handles this signal by change the p2_started flag. This flag used in the P1 source code to avoid the send the signal before P2 starts.
    - **SIGUSR2** : sent by P2 to P1. In the end of the P1, P1 must send “I am done” signal to P2. But what happens P2 started ? So, P1 must wait(suspend) until P2 starts. Then P1 can send “I am done signal” to P2. This **SIGUSR2** signal blocked in the P1 except sigsuspend state, so P1 can listen **SIGUSR2** just in the suspend state, otherwise signal can be lost.

**3) Handling the critical section**

Both processes have critical sections, and these sections should not be blocked. In order to do this, we can user **<signal>** API here too. **SIGINT** and **SIGSTOP** should be blocked in the critical section. So we create new signal mask by type sigset_t, add these signals into to sigset and change the processes current signal mask by this signal mask using ***sigprocmask()***. Now, process can not be interrupted by these signals. After end of the critical section, remove these signals from the signal mask set and change the processes signal mask by ***sigprocmask()*** again.

**Note : SIGSTOP cannot** be ignored or caught or handled. So the request that block the SIGSTOP in the code, will be ignored by the kernel. This request does nothing.