# SDP 2016/17 - Lab 2 - Davide Gallitelli S241521

## E01 - Sempahores and Signals

The goal of this exercise is to use signals in order to wake a process which is waiting on a semaphore.

In order to do so, an alarm handler is defined, in order to manage the SIG_ALRM received:

```
void ALRM_handler() {
    /* Called when SIG_ALRM is received */
    flag = 0;
    sem_post(S);
};
```

What it does is that a global flag is set to 1. The values for this flag are: - 0 - stands for "the program returns normally" - which means it has received a *sem_post* - 1 - default, stands for "the program returns for timeout"

The *wait_with_timeout* function sets an alarm (SIG_ALRM signal) for *tmax* seconds. To ensure maximum precision, *ualarm()* is used for smaller values of tmax (microseconds, the max argument allowed by *ualarm()* is one million microseconds) and a standard *alarm()* otherwise. After setting the alarm, the process waits on the semaphore S.

```
int wait_with_timeout(sem_t *S, int tmax) {
    flag=1;
    if (tmax < 1000)
        ualarm(tmax*1000,0);
    else
        alarm((float) tmax/1000);
    sem_wait(S);
    alarm(0);
    return flag;
};
```

If a sem_post is received before the timeout, the process itself will cancel all pending alarms, and it returns the default flag. Otherwise, the alarm goes off, generating a *SIG_ALRM* signal, which triggers the signal catcher. The flag is flipped, and returned to the thread A function.

```
[...]
result = wait_with_timeout(S, tmax);
    if (result == 1)
        printf("Wait returned normally. \n");
    else
        printf("Wait on semaphore S returned for timeout. \n");
[...]
```