SEMAPHORES

Semaphores as synchronization mechanisms are used to protect critical sections or to control limited resources (resource counters). Basic operations can be performed with functions:

```
int sem_init(sem_t *sem, int sync_proc, unsigned int initial_value);
int sem_set(sem_t *sem);
int sem_wait(sem_t *sem);
```

where `sem` is a pointer to a `sem_t` object, `sync_proc` is set to zero if the semaphore is used by threads within the same process, to a different value otherwise, and `initial_value` is the initial value for the semaphore.

When a semaphore is used to synchronize threads within the same process, the semaphore object is usually defined as a global variable, e.g. `sem_t sem;` and a pointer to it (`&sem`) is passed to semaphore functions (`sem_wait(&sem)`).

When a semaphore is used to synchronize threads from different processes, the semaphore object must be allocated in shared memory. In such scenarios, the parent process usually first creates shared memory, initializes a semaphore object in it, and then creates child processes that synchronize with that semaphore.

A sketch of code for both cases follow.

Threads within the same process:

```
...
sem_t sem; //global variable, outside functions
...
in function main:
      ...
      sem_init(&sem, 0, 1);
      ...
      pthread_create(...)
      ...

in threads:
      ...
      sem_wait(&sem);
      ...
      sem_post(&sem);
      ...
```

Threads within different processes:

```
...
sem_t *sem; //global variable, outside functions
...
in function main:
      ...
      allocate shared memory
      sem = address of share memory
      sem_init(sem, 1, 1);
      ...
      fork(...)
      ...

in child processes:
      ...
      sem_wait(sem);
      ...
      sem_post(sem);
      ...
```