semaphores

- controlled access to a counter
- can be used as a resource counter
- **two operations supported**:
  - `wait()`: decrement semaphore value if its values is positive, and continue; block calling process/thread otherwise
  - `post()`: increment semaphore value; if there was any process/thread blocked due to the semaphore, unblock one of them

System V semaphores

characteristics

- works with semaphore arrays
- may block a process and all the threads in it (**it is NOT thread safe**)

SystemV functions

```
int semget(key_t key, int nsems, int semflg);
// create a set of semaphores
```

```
int semctl(int semid, int semnum, int cmd, ...);
// semaphore control operations
// may have 3 or 4 arguments
```

```
int semop(int semid, struct sembuf *sops, unsigned nsops);
int semtimedop(int semid, struct sembuf *sops, unsigned nsops, struct timespec
*timeout);
// semaphore operations
```

Note

semlib library simplifies the use of System V semaphores!

POSIX semaphores

characteristics

- **thread safe**
- **two variants**:
    - named semaphores
    - unnamed semaphores

basic functions

```
int sem_post(sem_t *sem);
// increment the value of the semaphore
```

```
int sem_wait(sem_t *sem);
// decrement the value of the semaphore
// if semaphore value is not positive, will be blocked until it becomes positive
```

```
int sem_trywait(sem_t *sem);
// attempt to decrement the value of the semaphore without blocking
// if the semaphore value is greater than zero, it will decrement and return
immediately
// otherwise, it returns an error without blocking
```

```
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout;
// this function tries to decrement the semaphore but it waits only up to a
specified timeout
```

```
int sem_getvalue(sem_t *sem, int *sval);
// get the current value of the semaphore
```

Info

these functions return 0 on success, or -1 on error

unnamed semaphores

- must use shared memory for inter-process synchronization or internal memory for inter-thread synchronization

functions

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
// creation of an unnamed semaphore
```

```
- sem: semaphore is initialized at the address pointed by sem
- pshared: specifies if the semaphore will be sahred between threads in a process
(0) or between processes (1)
- value: initial value for the semaphore
```

```
int sem_destroy(sem_t *sem);
// destroy an unnamed semaphore
```

named semaphores

```
sem_t *sem_open(const char *name, int oflag);
// create an unnamed semaphore
```

```
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
// create an unnamed semaphore
```

```
- name: semaphore name
- oflag: O_CREAT | O_EXCL
- mode: permissions (in octal)
- value: initial value
```

```c
int sem_close(sem_t *sem);
// close a named semaphore (remove association with a semaphore)
```

```c
int sem_unlink(const char *name);
// delete a named semaphore
```