POSIX Semaphores

```
#include <semaphore.h>
```

- `sem_init`, `sem_destroy`, `sem_post`, `sem_wait`, `sem_trywait`

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- The above initializes a semaphore.
- Compile either with `-lrt` or `-lpthread`
- `pshared` indicates whether this semaphore is to be shared between the threads of a process, or between processes:
  - zero: semaphore is shared between the **threads of a process**; should be located at an address visible to **all threads**.
  - non-zero: semaphore is shared **among processes**.

POSIX Semaphore Operations

- `sem_wait()`, `sem_trywait()`
  - ```
    int sem_wait(sem_t *sem);
    ```
  - ```
    int sem_trywait(sem_t *sem);
    ```
  - Perform $\mathrm{P}(\mathrm{s})$ operation.
  - `sem_wait` blocks; `sem_trywait` will fail rather than block.
- `sem_post()`
  - ```
    int sem_post(sem_t *sem)
    ```
  - Performs $\mathrm{V}(\mathrm{s})$ operation.
- `sem_destroy()`
  - ```
    int sem_destroy(sem_t *sem);
    ```
  - Destroys a semaphore.

Creating and using a POSIX Semaphore

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>

extern int errno;

int main(int argc, char **argv)
{
    sem_t sp; int retval;

    /* Initialize the semaphore. */
    retval = sem_init(&sp,1,2);
    if (retval != 0) {
        perror("Couldn't initialize."); exit(3); }

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n",retval); getchar();

    sem_destroy(&sp);
    return 0;
}
```

Executing the Program

```
antoulas@sazerac:~/src/PosixSems$ ./semtest
Did trywait. Returned 0 >

Did trywait. Returned 0 >

Did trywait. Returned -1 >

antoulas@sazerac:~/src/PosixSems$
```

Initialize and Open a **named** *Semaphore*

```
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,
                mode_t mode, unsigned int value);
```

- creates a new POSIX semaphore OR opens an existing semaphore whose name is `name`.
- `oflag` specifies flags that control the operation of the call
  - `O_CREAT` creates the semaphore;
  - provided that both `O_CREAT` and `O_EXCL` are specified, an `error` is returned if a semaphore with `name` already exists.
- if `oflag` is `O_CREAT` then 2 more arguments have to be used:
  - `mode` specifies the permissions to be placed on the new semaphore.
  - `value` specifies the initial value for the new semaphore.

More on Named POSIX Semaphores

- A named semaphore is identified by a (persistent) name that has the form `/this_is_a_sample_named_semaphore`.
  - consists of an initial slash followed by a (large) number of character (but no slashes).
- If you want to "see" (list) all named sempahores in your (Linux) system look at directory `/dev/shm`

More on Named POSIX Semaphores

```
int sem_close(sem_t *sem)
```

- closes the named semaphore referred to by *sem* freeing the system resources the invoking process has used.

```
int sem_unlink(const char *name)
```

- removes the named semaphore in question.

```
int sem_getvalue(sem_t *sem, int *sval)
```

- obtains the current value of semaphore..
- the cheater API-call!

Named POSIX Semaphore

```
#include        <stdio.h>
...
#include        <sys/stat.h>
#include        <semaphore.h>

int main(int argc, char *argv[]){
const char *semname;
int op=0; int val=0;

if (argc==3)  {
        semname=argv[1]; op=atoi(argv[2]);
        }
else    {
        printf("usage: nameSem nameOfSem Operation\n"); exit(1);
        }

sem_t *sem=sem_open(semname, O_CREAT|O_EXCL, S_IRUSR|S_IWUSR, 0);

if (sem!= SEM_FAILED)
        printf("created new semaphore!\n");
else if (errno== EEXIST ) {
        printf("semaphore appears to exist already!\n");
        sem = sem_open(semname, 0);
        }
else ;

assert(sem != SEM_FAILED);
sem_getvalue(sem, &val);
printf("semaphore's before action value is %d\n",val);
```

Named Posix Semaphore

```
if ( op == 1 ) {
        printf("incrementing semaphore\n");
        sem_post(sem);
        }
else if ( op == -1 ) {
        printf("decrementing semaphore\n");
        sem_wait(sem);
        }
else if ( op == 2 ){
        printf("clearing up named semaphore\n");
        sem_close(sem);  // close the sem
        sem_unlink(semname); // remove it from system
        exit(1);
        }
else    printf("not defined operation! \n");

sem_getvalue(sem, &val);
printf("semaphore's current value is %d\n",val);
sem_close(sem);
return(0);
}
```

Execution Outcome

```
antoulas@sazerac:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas 1
created new semaphore!
semaphore's before action value is 0
incrementing semaphore
semaphore's current value is 1
antoulas@sazerac:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935  sem.antoulas
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas -1
semaphore appears to exist already!
semaphore's before action value is 1
decrementing semaphore
semaphore's current value is 0
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas 2
semaphore appears to exist already!
semaphore's before action value is 0
clearing up named semaphore
antoulas@sazerac:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas 1
created new semaphore!
semaphore's before action value is 0
incrementing semaphore
semaphore's current value is 1
```

Execution Outcome

```
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas 1
semaphore appears to exist already!
semaphore's before action value is 1
incrementing semaphore
semaphore's current value is 2
antoulas@sazerac:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935  sem.antoulas
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas -1
semaphore appears to exist already!
semaphore's before action value is 2
decrementing semaphore
semaphore's current value is 1
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas -1
semaphore appears to exist already!
semaphore's before action value is 1
decrementing semaphore
semaphore's current value is 0
antoulas@sazerac:~/PosixSems$ ./namedSem /antoulas 2
semaphore appears to exist already!
semaphore's before action value is 0
clearing up named semaphore
antoulas@sazerac:~/PosixSems$ ls /dev/shm/
pulse-shm-1024070233  pulse-shm-1294442337  pulse-shm-2927836935
pulse-shm-1274848112  pulse-shm-2305588894  pulse-shm-3888866544
antoulas@sazerac:~/PosixSems$
```

Locking a file

- Imposing read/write locks on files (or sections of files) is essential at times.

- ```
  #include <fnctl.h>

  int fnctl(int filedes, int cmd, struct flock *ldata)
  ```

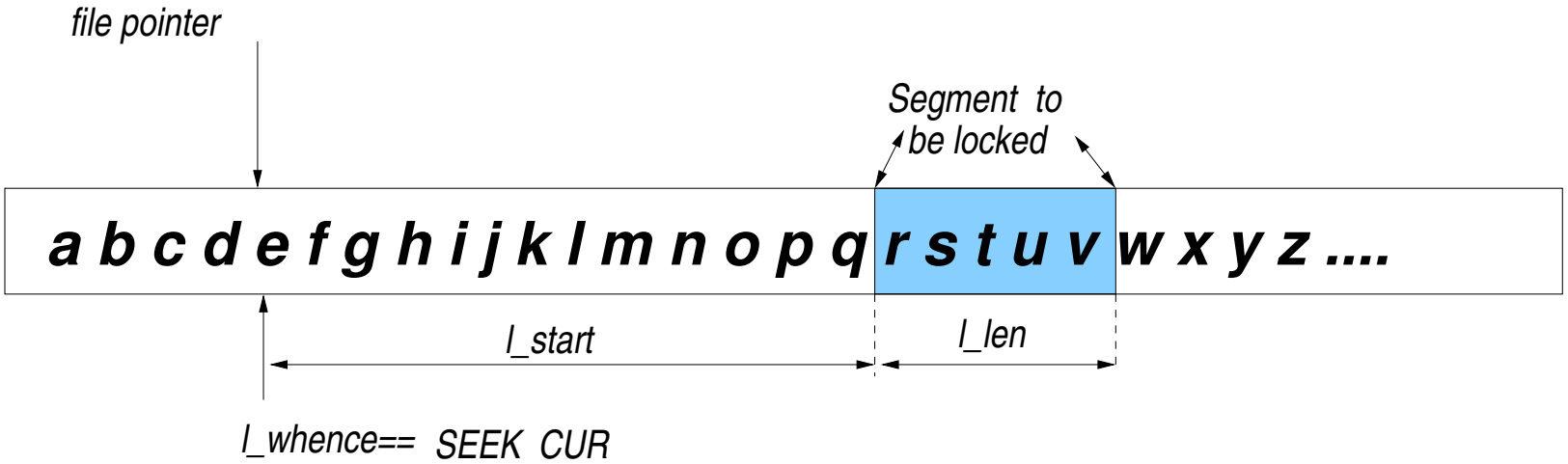
- File `filedes` must be opened with `O_RDONLY` or `O_WRONLY`.
- The `cmd` can be one of the three ("advisory locking"):
  - `F_GETLK`: get lock from data returned from `ldata`
  - `F_SETLK`: apply lock to a file; *return immediately* if this is not feasible.
  - `F_SETLKW`: apply lock to a file. However *wait*, if lock is blocked by a previous lock owned by another process.
  ```

The flock structure

- The `flock` structure is defined in `<fnctl.h>` and includes:

```
struct flock {
    ...
    short l_type;    /* Type of lock: F_RDLCK, F_WRLCK,
                        F_UNLCK                          */
    short l_whence;  /* How to interpret l_start: SEEK_SET,
                        SEEK_CUR, SEEK_END               */
    off_t l_start;   /* Starting offset for lock         */
    off_t l_len;     /* Number of bytes to lock          */
    pid_t l_pid;     /* PID of process blocking
                        our lock (F_GETLK only)          */
    ...
};
```

Locking a file



- `l_whence`: can be `SEEK_SET`, `SEEK_CUR` or `SEEK_END`.
  `l_start`: start position of the segment.
  `l_len`: segment in bytes.
- The `l_type` (lock type) can be:
  - `F_RDLCK`: lock to be applied is *read*
  - `F_WRLCK`: lock to be applied is *write*
  - `F_UNLCK`: lock on specified segment to be removed.

Locking a file

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main( ){
    int fd;
    struct flock my_lock;

    my_lock.l_type = F_WRLCK;
    my_lock.l_whence =  SEEK_SET;
    my_lock.l_start = 0 ;
    my_lock.l_len= 10;

    fd=open("locktest", O_RDWR);

    // lock first 10 bytes
    if ( fcntl(fd, F_SETLKW, &my_lock) == -1 ){
        perror("parent: locking");
            exit(1);
            }

    printf("parent: locked record \n");
```

Locking a file

```
    switch(fork()){
        case -1:
            perror("fork"); exit(1);
        case 0:
            printf("child: trying to lock file \n");
            my_lock.l_len = 5 ;
            if ( (fcntl(fd, F_SETLKW, &my_lock)) == -1 ){
                perror("child: problem in locking");
                exit(1);
                }
            printf("child: locked \n"); sleep(1);
            printf("child: exiting \n");
            fflush(stdout); fflush(stderr); exit(1);
        default:
            printf("parent: just about unlocking now \n");
            sleep(5);
            my_lock.l_type = F_UNLCK;
            printf("parent: unlocking -now- \n");
            if ( fcntl(fd, F_SETLK, &my_lock) == -1 ){
                perror("parent: problem in unlocking! \n");
                exit(1); }
            printf("parent: has unlocked and is now exiting \n");
            fflush(stdout); fflush(stderr); wait(NULL);
    }
    sleep(2);
}
```

Execution Outcome

```
antoulas@sazerac:~/Filelocking$ ./lockit
parent: locked record
child: trying to lock file
parent: just about unlocking now
parent: unlocking -now-
parent: has unlocked and is now exiting
child: locked
child: exiting
antoulas@sazerac:~/Filelocking$
```

Possible Deadlock

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main( ){
    int fd;
    struct flock first_lock;
    struct flock second_lock;

    first_lock.l_type = F_WRLCK;
    first_lock.l_whence =  SEEK_SET;
    first_lock.l_start = 0 ;
    first_lock.l_len= 10;

    second_lock.l_type = F_WRLCK;
    second_lock.l_whence =  SEEK_SET;
    second_lock.l_start = 10;
    second_lock.l_len= 5;


    fd=open("locktest", O_RDWR);

    if ( fcntl(fd, F_SETLKW, &first_lock) == -1 )
        perror("-A:");
    printf("A: lock obtained by processs %d \n",getpid());

    switch(fork()) {
        case -1:
            perror("error on fork");
            exit(1);
```

Possible Deadlock

```
        case 0: /* child */
            if ( fcntl(fd, F_SETLKW, &second_lock) == -1 )
                perror("-B:");
            printf("B: lock obtained by process %d\n",getpid());

            if ( fcntl(fd, F_SETLKW, &first_lock) == -1 ){
                perror("-C:");
                printf("Process %d terminating\n",getpid());
                     exit(1);
                }
            else printf("C: lock obtained by process %d\n",getpid());
            printf("Process %d successfully acquired BOTH locks \n",getpid());
            exit(0);
        default: /* parent */
            printf("Parent process %d sleeping \n",getpid());
            sleep(10);
            if ( fcntl(fd, F_SETLK, &second_lock) == -1 ){
                perror("--D:");
                printf("Process %d about to terminate\n",getpid());
                }
            else printf("D: lock obtained by process %d\n",getpid());
            sleep(1);
            printf("Process %d on its way out of here \n",getpid());
    }
}
```

Execution Outcome

```
antoulas@sazerac:~/Filelocking$ ./deadlock
A: lock obtained by processs 10822
Parent process 10822 sleeping
B: lock obtained by process 10823
--D:: Resource temporarily unavailable
Process 10822 about to terminate
Process 10822 on its way out of here
C: lock obtained by process 10823
Process 10823 successfully acquired BOTH locks
antoulas@sazerac:~/Filelocking$
```