

HOW YOUR PROGRAM GENERATES RANDOM NUMBERS

- Generating Random Numbers:

You need to include `<stdlib.h>`:

For this assignment, you must initialize the random

stream using: `long seed = 8 ;`

and:

`srand((unsigned) seed) ;` //this must be done before generating
any numbers

HOW YOUR PROGRAM GENERATES RANDOM NUMBERS

to generate the stream of numbers, you use

`rand() % 256` for each digit

for example, if you wanted to fill an array called `RANDOM` with 100 random digits between 0 and 255 you would use the following:

```
for (i = 0 ; i < 100 ; i++)  
    RANDOM[i] = rand % 256 ;
```

note that we will be generating a $1024 * 1024$ array of random numbers between 0 – 255.

USING ONE SEMAPHORE WITH PTHREADS

To use semaphores, you must include the following header files:

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

You need to create a name for your semaphore such as:

```
char *semname = "sema" ;
```

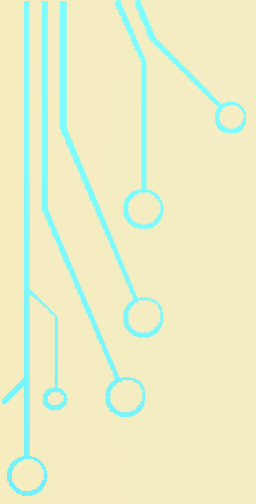
And declare a variable of type sem_t:

```
sem_t *my_sem ;
```

you create/initialize the semaphore using:

```
my_sem = sem_open(semname, O_CREAT, 777, 1) ;
```

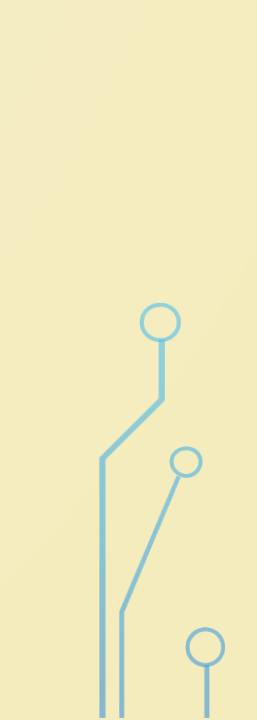
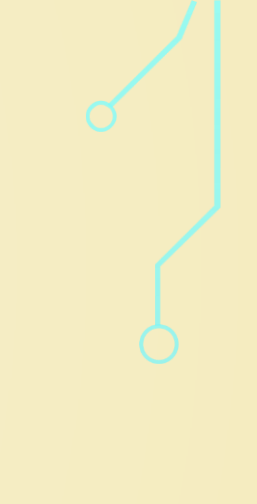
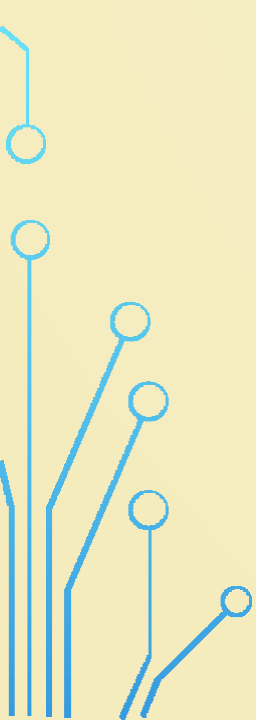
which creates a semaphore called my_sem and initializes its count variable to 1. Don't worry about the 777 yet.



You wait on the semaphore using: `sem_wait(my_sem) ;`

and signal the semaphore using: `sem_post(my_sem) ;`

Note that you do not use the address of the semaphore.



Then to use the semaphore for protection of a critical section (using the simple Counter++/-- example in a thread we have been using:

```
void *inc(void * hubb)
{
    sem_wait(my_sem) ;
    counter++ ;
    sem_post(my_sem);
    return (void *) NULL; }
```

```
void *dec(void * bub)
{
    sem_wait(my_sem) ;
    counter-- ;
    sem_post(my_sem);
    return (void *) NULL; }
```

CRITICAL

- Before the program exits you must close the semaphore and unlink it.

```
sem_close(my_sem) ;
```

```
sem_unlink(semname) ;
```

Assume we are using an array of 10 semaphores rather than 256.

```
char *sems[10] = {"S0", "S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9"} ;  
  
sem_t *my_sem_array[10] ;
```

To create/initialize the array of semaphores:

```
for(i = 0; i < 10 ; i++)  
    my_sem_array[i] = sem_open(sems[i], O_CREAT, 777, 1) ;
```

To access the critical sections, each thread would:

```
for(I = 0; I < 10 ; i++)  
  
    {sem_wait(my_sem_array[i])      ;  
      //update rand_digit[i] ;  
      sem_post(my_sem_array[i]) ;  
    }
```


CRITICAL!

You must close and unlink the all semaphores after finishing with the program. In the case of the array of semaphores, you must:


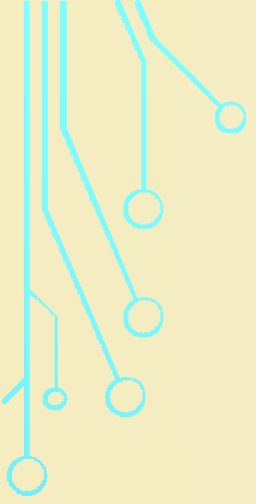
```
for(i = 0; i < 10 ; i++)  
{  
    sem_close(my_sem_array[i]) ;  
    sem_unlink(sems[i]) ;  
}
```

We are actually using an array of 256 semaphores.

```
char Sems[256][10] ;  
sem_t *My_Sems[256] ;
```

```
for(i = 0; i < 256 ; i++)  
    { sprintf(Sems[i], "S%d", i) ; //create array of name  
      My_Sems[i]=sem_open(Sems[i], O_CREAT,  
777,1);  
    }
```

Use My_Sems[i] to protect total_random[i] ;



```
for(i = 0; i < 256; i++)  
    {  
        sem_close(My_Sems[i]);  
        sem_unlink(Sems[i])    ;  
    }
```

