| **Operation** | **Destination Discussion** |
| --- | --- |
| Initialize a semaphore | [sem\_init(3RT)](https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnj/index.html#sync-19683) |
| Increment a semaphore | [sem\_post(3RT)](https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnj/index.html#sync-93728) |
| Block on a semaphore count | [sem\_wait(3RT)](https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnj/index.html#sync-91984) |
| Decrement a semaphore count | [sem\_trywait(3RT)](https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnj/index.html#sync-87966) |
| Destroy the semaphore state | [sem\_destroy(3RT)](https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnj/index.html#sync-59226) |

#include<semaphore.h>

**Initialization of Semaphore**

int = sem\_init(sem\_t \**sem*, int *pshared*, unsigned int *value*);

sem\_t some\_sem; //defining some semaphore

int pshared; // sharing among the process if 0 no sharing else shared

int err;

int value; // value can be any integer number

pshared = 0;

value = 1;

if (err = sem\_init(&sem, pshared, value)!=0)

printf(“error”);

else

printf(“successfully executed with return=%d”,err);

//

if return 0 returned successful execution of sem operation

**Use of Semaphore wait**

int sem\_wait(sem\_t \*sem);

#include <semaphore.h>

sem\_t some\_sem;

int err;

if(err= sem\_wait(&some\_sem)!=0) //wait for semaphore

printf(“error”)

else

printf(“successfully executed with return=%d”,err);

// if return 0 returned successful execution of sem operation

**Use of semaphore signal**

int sem\_post(sem\_t \*sem);

#include <semaphore.h>

sem\_t some\_sem;

int err;

if(err= sem\_post(&some\_sem)!=0) //signal a semaphore

printf(“error”)

else

printf(“successfully executed with return=%d”,err);

// if return 0 returned successful execution of sem operation

Example program to use the semaphore operations

#include <sys/types.h>

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include<semaphore.h>

void\* func1(void\* val);

void\* func2(void\* val);

int intg1 = 5; // shared variable1

sem\_t some\_sem1; // declare semaphore1

int main()

{

void\* status;

pthread\_t some\_thread1;

pthread\_t some\_thread2;

sem\_init(&some\_sem1, 0, 1); // initialize semaphores(some semaphore,no of process to be shared,count value)

pthread\_create(&some\_thread1, NULL, func1, NULL);

pthread\_create(&some\_thread2, NULL, func2, NULL);

pthread\_join(some\_thread1, &status);

pthread\_join(some\_thread2, &status);

sem\_destroy(&some\_sem1);

printf("The value of intg1 returned to main=%d",intg1);

return 0;

}

void \*func1(void \*val)

{

int x;

sem\_wait(&some\_sem1); // wait

x=intg1; // write

printf("The value of x = %d\n", x);

x+=1;

sleep(2);

intg1=x;

printf("The value of intg1 in func1= %d\n", intg1); // read

sem\_post(&some\_sem1); // signal

}

void \*func2(void \*val)

{

int y;

sem\_wait(&some\_sem1);//wait

y=intg1;

//sem\_wait(&some\_sem1); // wait

printf("The vlaue of y = %d\n", y); // read

y-=1;

sleep(2);

intg1=y; // write

printf("The vlaue of intg1 in func2 = %d\n", intg1);

sem\_post(&some\_sem1); // signal

}