# EXPERIMENT 10 – SEMAPHORES

##### OBJECTIVES

* Solving critical section problem using semaphores.
* Semaphore implementation.

**TIME REQUIRED** : 3 hrs

**PROGRAMMING LANGUAGE** : C/C++/Java

**SOFTWARE REQUIRED** : Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans

**HARDWARE REQUIRED** : Core i5 in Computer Labs

##### SEMAPHORES

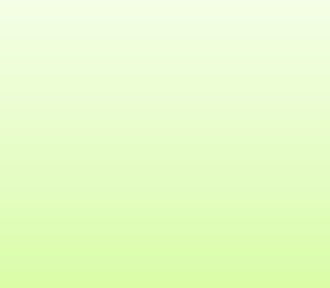
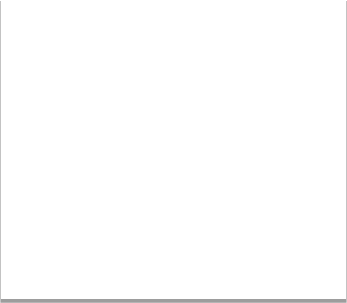
In [computer science,](https://en.wikipedia.org/wiki/Computer_science) a semaphore is a [variable](https://en.wikipedia.org/wiki/Variable_(programming)) or [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) used to control access to a common resource by multiple [processes](https://en.wikipedia.org/wiki/Process_(computing)) in a [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing) system such as a [multitasking](https://en.wikipedia.org/wiki/Computer_multitasking) operating system. A semaphore is simply a variable. This variable is used to solve [critical section](https://en.wikipedia.org/wiki/Critical_section) problems and to achieve process synchronization in the multi-processing environment. A trivial semaphore is a plain variable that is changed (for example, incremented or decremented, or toggled) depending on programmer-defined conditions.

A useful way to think of a semaphore as used in the real-world system is as a record of how many units of a particular resource are available, coupled with operations to adjust that record safely (i.e., to avoid [race](https://en.wikipedia.org/wiki/Race_condition) [conditions](https://en.wikipedia.org/wiki/Race_condition)) as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores and are used to implement [locks](https://en.wikipedia.org/wiki/Lock_(computer_science)).

The semaphore concept was invented by [Dutch](https://en.wikipedia.org/wiki/Dutch_people) [computer scientist](https://en.wikipedia.org/wiki/Computer_scientist) [Edsger Dijkstra](https://en.wikipedia.org/wiki/Edsger_Dijkstra) in 1962 or 1963, when Dijkstra and his team were developing an [operating system](https://en.wikipedia.org/wiki/Operating_system) for the [Electrologica X8](https://en.wikipedia.org/wiki/Electrologica_X8). That system eventually became known as [THE multiprogramming system](https://en.wikipedia.org/wiki/THE_multiprogramming_system).

An integer value used for signalling among processes. Only three operations may be performed on a semaphore, all of which are atomic:

* Initialize,
* Decrement (Wait)
* Increment. (Signal)



wait (S){

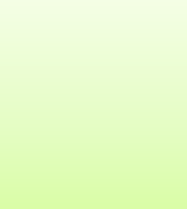
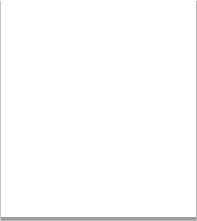
while (S <= 0){

; // wait

}

S--;

}



signal

(S) {

S++;

}

When one process modifies semaphore value, no other process can simultaneously modify that semaphore. When a process releases resource, it performs signal. When count goes to 0, all resources are being used. After that, process that wish to use a resource will block until count becomes greater than 0.

##### COUNTING SEMAPHORE:

Integer value can range over an unrestricted domain. Counting semaphores can be used to control access to a given resource with finite number of instances.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#define KEY 0x1111

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*array;

};

struct sembuf p = { 0, -1, SEM\_UNDO};

struct sembuf v = { 0, +1, SEM\_UNDO};

int main()

{

int id = semget(KEY, 1, 0666 | IPC\_CREAT);

if(id < 0)

{

perror("semget"); exit(11);

}

union semun u;

u.val = 1;

if(semctl(id, 0, SETVAL, u) < 0)

{

perror("semctl"); exit(12);

}

int pid;

pid = fork();

srand(pid);

if(pid < 0)

{

perror("fork"); exit(1);

}

else if(pid)

{

char \*s = "abcdefgh";

int l = strlen(s);

for(int i = 0; i < l; ++i)

{

if(semop(id, &p, 1) < 0)

{

perror("semop p"); exit(13);

}

putchar(s[i]);

fflush(stdout);

sleep(rand() % 2);

putchar(s[i]);

fflush(stdout);

if(semop(id, &v, 1) < 0)

{

perror("semop p"); exit(14);

}

sleep(rand() % 2);

}

}

else

{

char \*s = "ABCDEFGH";

int l = strlen(s);

for(int i = 0; i < l; ++i)

{

if(semop(id, &p, 1) < 0)

{

perror("semop p"); exit(15);

}

putchar(s[i]);

fflush(stdout);

sleep(rand() % 2);

putchar(s[i]);

fflush(stdout);

if(semop(id, &v, 1) < 0)

{

perror("semop p"); exit(16);

}

sleep(rand() % 2);

}

}

}