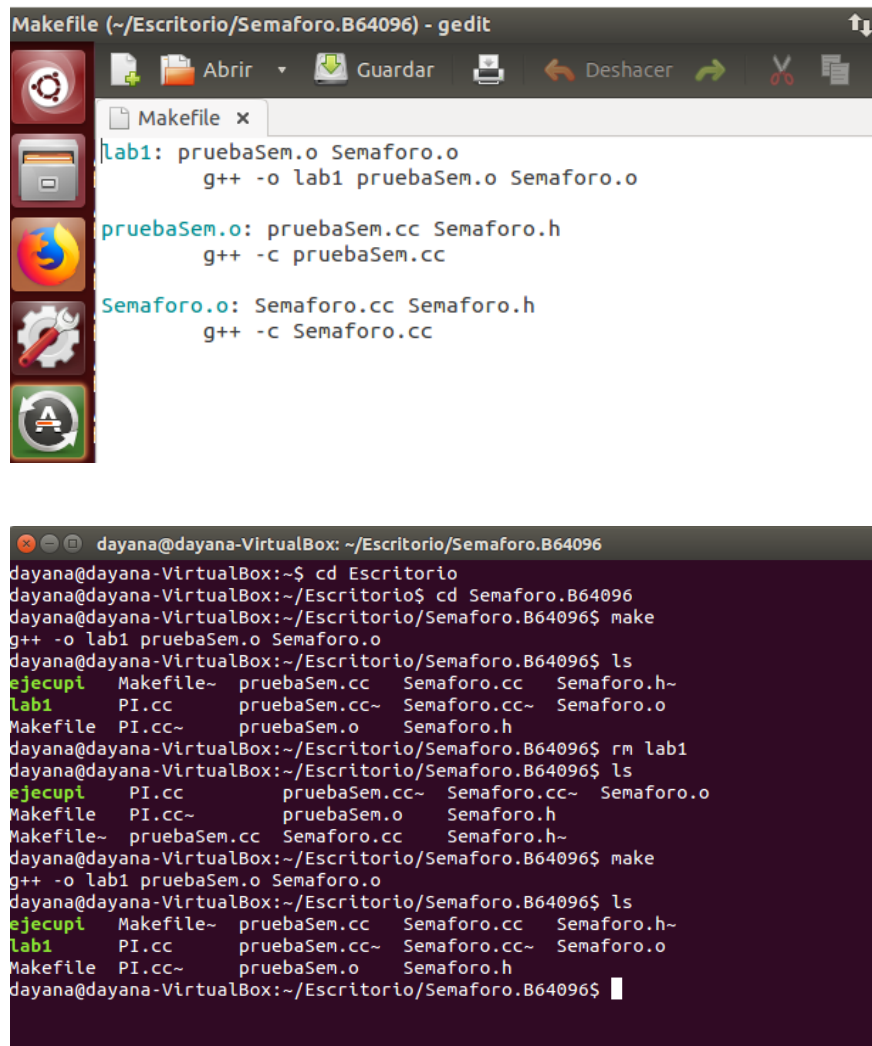


1. [Entender como funciona los Makefiles]



The image shows two screenshots. The top screenshot is a gedit window titled 'Makefile (~/Escritorio/Semaforo.B64096) - gedit'. It contains the following Makefile content:

```
lab1: pruebaSem.o Semaforo.o
    g++ -o lab1 pruebaSem.o Semaforo.o

pruebaSem.o: pruebaSem.cc Semaforo.h
    g++ -c pruebaSem.cc

Semaforo.o: Semaforo.cc Semaforo.h
    g++ -c Semaforo.cc
```

The bottom screenshot is a terminal window titled 'dayana@dayana-VirtualBox: ~/Escritorio/Semaforo.B64096'. It shows the following commands and output:

```
dayana@dayana-VirtualBox:~/Escritorio$ cd Semaforo.B64096
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ make
g++ -o lab1 pruebaSem.o Semaforo.o
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ls
ejecupi Makefile~ pruebaSem.cc Semaforo.cc Semaforo.h~
lab1 PI.cc pruebaSem.cc~ Semaforo.cc~ Semaforo.o
Makefile PI.cc~ pruebaSem.o Semaforo.h
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ rm lab1
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ls
ejecupi PI.cc pruebaSem.cc~ Semaforo.cc~ Semaforo.o
Makefile PI.cc~ pruebaSem.o Semaforo.h
Makefile~ pruebaSem.cc Semaforo.cc Semaforo.h~
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ make
g++ -o lab1 pruebaSem.o Semaforo.o
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ls
ejecupi Makefile~ pruebaSem.cc Semaforo.cc Semaforo.h~
lab1 PI.cc pruebaSem.cc~ Semaforo.cc~ Semaforo.o
Makefile PI.cc~ pruebaSem.o Semaforo.h
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$
```

2. [Creación de procesos en Unix (Linux) [fork] Correr el ejemplo provisto "Pi-PorSeries.c" Crear un documento de texto y anotar los resultados obtenidos]

- La razón por la que no funciona el código de PI es que el programa al crear el fork genera un proceso hijo, este tiene su vector propio así como su inicio y su final, esto hace que cuando se calcule la suma y almacene el resultado en el vector dado no quede guardado en el vector global ya que el se destruye, al morir el proceso hijo no queda nada guardado en el programa, ya que cada uno de los procesos del 1 al 10 se destruyen sin enviar ninguna señal al proceso padre o en sí al programa.

```

dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ g++ -o ejecupi PI.cc
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ls
ejecupi  Makefile~  pruebaSem.cc  Semaforo.cc  Semaforo.h~
lab1     PI.cc      pruebaSem.cc~  Semaforo.cc~  Semaforo.o
Makefile PI.cc~     pruebaSem.o    Semaforo.h
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ./ejecupi
Creating process 5012: starting value 0, finish at 100000
Creating process 5013: starting value 100000, finish at 200000
Creating process 5014: starting value 200000, finish at 300000
Creating process 5015: starting value 300000, finish at 400000
Creating process 5016: starting value 400000, finish at 500000
Creating process 5017: starting value 500000, finish at 600000
Creating process 5018: starting value 600000, finish at 700000
Creating process 5019: starting value 700000, finish at 800000
Creating process 5020: starting value 800000, finish at 900000
Creating process 5021: starting value 900000, finish at 1000000
Valor calculado de Pi es 0 con 1000000 terminos
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$

```

### 3. [Semáforos: construir la implantación de la clase]

- Para efectos de la solución del punto c del laboratorio el profesor otorgó dos archivos los cuales son: Semaforo.h y pruebaSem.cc. Debido esto se trabajó en un único archivo el cual es Semaforo.cc de la siguiente manera:
- En primera parte se definió la llave para el constructor del semáforo, se declaran las librerías correspondientes y se crea una union necesaria para el constructor ya que contiene el valor que se le asigna al semáforo.

```

#define KEY 0xB64096
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include "Semaforo.h"

//g++ Semaforo.cc pruebaSem.cc

union semun {
    int val = 0;    /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* Array for GETALL, SETALL */
    struct seminfo *__buf; /* Buffer for IPC_INFO
                          /* (Linux-specific) */
};
struct sembuf l;
union semun u;

```

- Después se crea el constructor del semáforo con sus respectiva ID, demás parámetros y el destructor. Se utiliza semget que obtiene el identificador de un conjunto de semáforos.

```

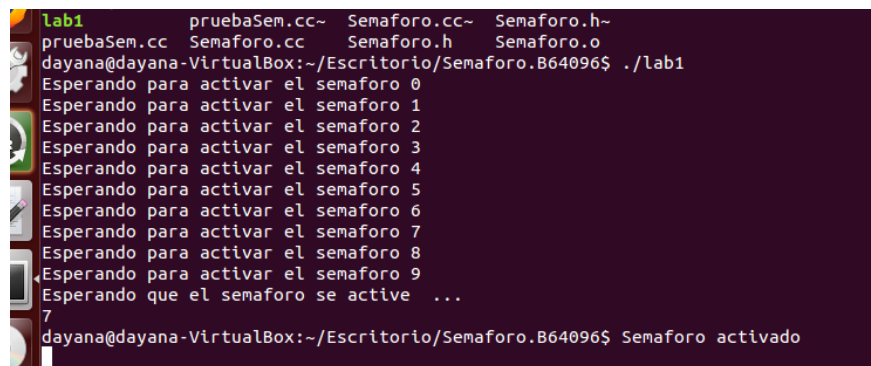
Semaforo::Semaforo(int inicial){
    int estado;
    this->id = semget(KEY,1,IPC_CREAT |0600);
    if(-1==id){
        perror("Semaforo::Semaforo");
        exit(1);
    }
    estado=semctl(id,0,SETVAL,u);
    if(-1==estado){
        perror("Semaforo::Semaforo");
        exit(1);
    }
}
Semaforo::~Semaforo(){
    semctl(id, 0, IPC_RMID);
}

```

- Se crea el metodo Wait y Signal los cuales indican a los procesos si deben esperar o continuar. Se utiliza semop que es para operaciones con semáforos. sem\_num indica el número de semáforo, sem\_op indica operacion sobre el semáforo y sem\_flg que indica baderas o indicadores para la operacion.

```
int Semaforo::Wait(){
    l.sem_num=0;
    l.sem_op=-1;
    l.sem_flg=0;
    int estado=semop(id,&l,1);
}
int Semaforo::Signal(){
    l.sem_num=0;
    l.sem_op=1;
    l.sem_flg=0;
    int estado =semop(id,&l,1);
}
```

- Al compilar el programa genera la solución esperada.



```
lab1      pruebaSem.cc~ Semaforo.cc~ Semaforo.h~
pruebaSem.cc Semaforo.cc Semaforo.h Semaforo.o
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ ./lab1
Esperando para activar el semaforo 0
Esperando para activar el semaforo 1
Esperando para activar el semaforo 2
Esperando para activar el semaforo 3
Esperando para activar el semaforo 4
Esperando para activar el semaforo 5
Esperando para activar el semaforo 6
Esperando para activar el semaforo 7
Esperando para activar el semaforo 8
Esperando para activar el semaforo 9
Esperando que el semaforo se active ...
7
dayana@dayana-VirtualBox:~/Escritorio/Semaforo.B64096$ Semaforo activado
```

⋮