

# Reporte Final

Sistemas Operativos

Ana Paola Nava Vivas & Erick Gabriel Mendoza  
Flores

ESCUELA SUPERIOR DE CÓMPUTO

## INTRODUCCIÓN

El proyecto final de Sistemas Operativos consiste, como es de esperarse, de un sistema conjunto, conformado por diferentes componentes de software ejecutados de manera sincrónica para transportar datos de un lado a otro. Se aplicaron diversas técnicas aprendidas en clase. Las más prominentes siendo el uso de hilos para el multiprocesamiento y la implementación de semáforos y memoria compartida.

## ESTRUCTURA

El proyecto consta de tres programas que realizan diferentes funciones. Para ejecutarlos se requirió utilizar dos computadoras conectadas a una red local. Los equipos utilizados utilizaron sistemas operativos de la familia Linux. La especificación a grandes rasgos es la siguiente:

Computadora 1:

- ❖ Ubuntu Mate (GNU/Linux).
- ❖ Procesador i5 3ra Generación.
- ❖ Productor – Consumidor / Cliente

Computadora 2:

- ❖ Debian Sid (GNU/Linux).
- ❖ Procesador Atom N450.
- ❖ Servidor.

Se hizo uso de herramientas y APIs del estándar IEEE Std 1003.1-2008 para portabilidad entre sistemas operativo. La principal de su implementación fue la facilidad de uso y documentación detallada. La memoria compartida consistió en el uso de una estructura sencilla que contiene un arreglo bidimensional de caracteres que es la zona crítica.

COMPUTADORA 1:

Ejecutó un programa productor y un programa consumidor/cliente. Ambos programas utilizaron un espacio de memoria compartida sobre el cuál escribieron 3 caracteres que identificaban el tipo de mensaje que compartieron entre sí. Así mismo, los programas compartieron 8 semáforos en total, que eran controlados de forma cruzada, es decir, el productor controlaba el incremento de los semáforos donde se acumulan los hilos del programa consumidor y el decremento de los propios. De igual manera, funcionaba el proceso inverso con el consumidor. El productor constó de 3 hilos y el consumidor de 2.

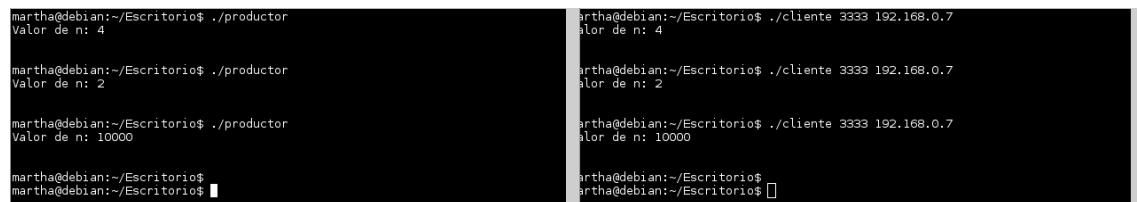
COMPUTADORA 2:

Conformó el servidor. El programa se encargó de recibir la información que era enviada por el programa cliente dentro de la computadora 1 y de almacenar dicha información de manera ordenada de su lado. Al recibir los datos, los clasificaba en ese mismo momento, separándolos por archivos distintos y en carpetas correspondientes.

## EJECUCIÓN

Los programas se ejecutaron un total de 3 veces para corroborar su funcionamiento. La primera vez se ejecutó con 2 iteraciones de parte de los productores y consumidores. La segunda se hizo con 4 y la tercera con 10,000. Se observó del lado del servidor que en 10,000 iteraciones ocurrieron pausas no muy prolongadas en la recepción de datos. Sin embargo, se entregaron los 90,000 paquetes de manera exitosa.

*Iteraciones (del lado del cliente):*



```
martha@debian:~/Escritorio$ ./productor
Valor de n: 4

martha@debian:~/Escritorio$ ./productor
Valor de n: 2

martha@debian:~/Escritorio$ ./productor
Valor de n: 10000

martha@debian:~/Escritorio$
martha@debian:~/Escritorio$
```

```
martha@debian:~/Escritorio$ ./cliente 3333 192.168.0.7
Valor de n: 4

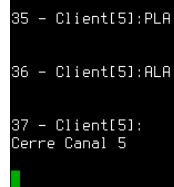
martha@debian:~/Escritorio$ ./cliente 3333 192.168.0.7
Valor de n: 2

martha@debian:~/Escritorio$ ./cliente 3333 192.168.0.7
Valor de n: 10000

martha@debian:~/Escritorio$
martha@debian:~/Escritorio$
```

Figura 1

*Iteraciones (del lado del servidor):*

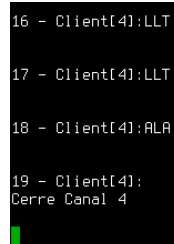


```
35 - Client[5]:PLA

36 - Client[5]:ALA

37 - Client[5]:
Cerre Canal 5
```

Figura 2



```
16 - Client[4]:LLT

17 - Client[4]:LLT

18 - Client[4]:ALA

19 - Client[4]:
Cerre Canal 4
```

Figura 3



```
89998 - Client[5]:LLT

89999 - Client[5]:LLA

90000 - Client[5]:LLM

90001 - Client[5]:
Cerre Canal 5
```

Figura 4

Se puede observar en todos los casos que la computadora 1 produce  $n$  iteraciones, la computadora 2 recibe 9 veces  $n$  datos. Esto es porque del lado del productor existen 3 hilos produciendo 3 datos, dando un total de 9 datos por hilo.

## PROGRAMAS

```
void dofile(FILE * file, char * path, char * flag, char * msg)
{
    file = fopen(path, flag); if(file==NULL){puts("Error with file");exit(0);}
    fprintf(file, "%c%c%c\n", msg[0], msg[1], msg[2]);
    fclose(file);
}

void archivito(char * msg)
{
    if(msg[0]=='A')
    {
        if(msg[1]=='M' && msg[2]=='A'){dofile(file_cntrl, "/ANA/m_att.txt", "a", msg);}
        else if(msg[1]=='M' && msg[2]=='M'){dofile(file_cntrl, "/ANA/m_mov.txt", "a", msg);}
        else if(msg[1]=='M' && msg[2]=='T'){dofile(file_cntrl, "/ANA/m_tel.txt", "a", msg);}
        else if(msg[1]=='L' && msg[2]=='A'){dofile(file_cntrl, "/ANA/l_att.txt", "a", msg);}
        else if(msg[1]=='L' && msg[2]=='M'){dofile(file_cntrl, "/ANA/l_mov.txt", "a", msg);}
        else if(msg[1]=='L' && msg[2]=='T'){dofile(file_cntrl, "/ANA/l_tel.txt", "a", msg);}
    }
}
```

Figura 5

El programa del servidor recibe los mensajes y los sortea en sus carpetas y con sus tipos correspondientes.

```
printf("La IP: %s El canal: %d\n", inet_ntoa(client.sin_addr), *id_canal);
while(1){
    if((no_bytes=recv(*(id_canal), (void *)buffer, sizeof(buffer), 0))==-1){
        printf("error al recibir\n");exit(1);}
    else{
        buffer[no_bytes]='\0';
        sem_wait(file_sem);
        archivito(buffer);
        sem_post(file_sem);
        cont++;
        printf("%d - Client[%d]:%s\n", cont, *id_canal, buffer);
    }
}
```

Figura 6

Hay un semáforo que sirve para controlar el ingreso a los archivos que debe hacerse de forma secuencial y uno detrás del otro. Esto permite hacer uso de un solo descriptor de archivos.

```
if(sem_getvalue(sem5, &valor)!=0) perror("error");
else{
    if(valor>0){
        sem_wait(sem5);
        sprintf(buffer, "%c%c%c", zonita->zona critica[0][0], zonita->zona critica[0][1], zonita->zona critica[0][2]);
        if(send(socket_id, (void *)buffer, strlen(buffer), 0)==-1){perror("error send");}
        if(recv(socket_id, (void *)buffer, sizeof(buffer), 0)==-1){perror("error recv");}
        sem_post(sem1);
        //if(!memcmp(buffer, "ok", 2))break;
    }
    else{
        if(sem_getvalue(sem6, &valor)!=0) perror("error");
        else{
            if(valor>0){
                sem_wait(sem6);
                sprintf(buffer, "%c%c%c", zonita->zona critica[1][0], zonita->zona critica[1][1], zonita->zona critica[1][2]);
                if(send(socket_id, (void *)buffer, strlen(buffer), 0)==-1){perror("error send");}
                if(recv(socket_id, (void *)buffer, sizeof(buffer), 0)==-1){perror("error recv");}
                sem_post(sem2);
                //if(!memcmp(buffer, "ok", 2))break;
            }
        }
    }
}
```

Figura 7

Los programas de productor y cliente son muy similares, la única diferencia es que el productor escribe en la zona crítica y el consumidor/cliente lee y envía la información.

## CONCLUSIÓN

El uso de tecnologías como los semáforos y los hilos no tienen mucho sentido al utilizarse de forma aislada. Uno pensaría que basta con tener un programa secuencial bien construido. Es cuando se implementan de forma conjunta, apoyándose de interfaces tan versátiles como los sockets u otras APIs, como las de interfaces gráficas, que podemos percatarnos de la utilidad de estas aplicaciones de software.

Todas estas propiedades del software también nos ayudan a comprender cómo es que escala un proyecto que, con suficientes módulos, se transforma en un sistema operativo y a medida que pasa el tiempo adquiere más y mejores funciones. La aplicación de sincronización de procesos es de gran importancia, ya que puede no implementarse correctamente, lo que llevaría a consecuencias graves en materia de seguridad y compromiso de memoria.

También hemos valorado la versatilidad del uso de software libre, ya que compartir programas y diseños ayudan a crecer a diversas comunidades y en turno, generar programas de mayor calidad con un soporte técnico que hasta ahora ninguna empresa privada puede lograr.