Primer Certamen
Tiempo 150 min. Al terminar, entregar sus respuestas vía AULA.

1.- (35 pts) Las grabaciones de las sesiones Zoom son almacenadas en un directorio por cada sesión zoom y en él se crea, entre otros, un archivo de audio (audio_only.m4a). Cree un script classd.sh que permita determinar las duración de la grabación de audio más corta de entre todas las grabaciones de una asignatura.

Descripción: classd.sh entrega la fecha y hora de la grabación más corta para las sesiones zoom grabadas en el directorio donde se ejecuta classd.sh

Ejemplo: suponiendo que tenemos grabaciones zoom en aragorn, estando en el directorio de grabaciones, entonces la ejecución y salida es como sigue:

agustin.gonzalez@aragorn:~$ ./classd.sh
Fecha grabación más corta: 2020-12-16 10.56.22 Duración: 3610.000 segundos

Hints: Para obtener la duración de un archivo de audio en segundos, se sugiere utilizar el utilitario ffprobe. Para obtener información sobre un archivo puede usar:

$ ffprobe -show_format audio_only.m4a

Para obtener solo la duración en segundos, use:

$ffprobe -show_entries stream=duration -v quiet -of default=noprint_wrappers=1:nokey=1 audio_only.m4a

Un ejemplo para la salida de este último comando es:

4875.104000

Zoom incluye la fecha de la grabación como parte del nombre del directorio de cada sesión. Ver: http://aragorn.elo.utfsm.cl/~agustin.gonzalez/elo330/Zoom/

R:

```
#! /bin/bash
min=0
for i in *.  # 5 pts.
do
 if  test -d "$i"
 then
          #echo "$i"
          duration=`ffprobe   -show_entries stream=duration -v quiet -of default=noprint_wrappers=1:nokey=1
"$i"/audio_only.m4a`.  # 5 pts
          duration=${duration%.*}   # o la línea siguiente para eliminar decimales.   5 pts.
   #duration=`echo $duration | cut -d "." -f 1` #  menos eficiente por crear procesos nuevos.
          if test $min -eq 0.     # 5 pts. Caso inicial
            then
               min=$duration
```

```
                session="$i"
        fi
        if test $duration -lt $min.   #  5pts caso no inicial
    then
            min=$duration
              session="$i"
  fi
 fi
done
set -- $session    # extracción de fecha 5 pts.
echo "Fecha de Grabación más corta: $1 $2   Duración: $min segundos".   # 5 pts.
```

2.- Se desea determinar cuán rápida es la transferencia de datos al usar FIFO respecto de usar memoria compartida.

a) (30 puntos) Cree un programa productor y otro consumidor. El primero transfiere, a través de un FIFO, N bytes al proceso consumidor. Cada char enviado contiene un número 1. El proceso consumidor es quien crea el FIFO, luego suma todos los N valores recibidos, muestra la suma de los datos recibidos y termina (debería imprimir N).

La ejecución del proceso Consumidor es:

$ consumidor <fifoname> <N>

La ejecución del proceso Productor es:

$ productor <fifoname> <N>

```
/* consumidor.c */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argvc, char * argv[]) {
    int n, fd;
    char data=1;
    char *fifoname=argv[1];
    int N = atoi(argv[2]);
    int sum = 0;
    /* Remove any previous FIFO.*/
    unlink(fifoname);

    /* Create the FIFO.  */
    if (mkfifo(fifoname, 0666) < 0) {
        perror("mkfifo");
        exit(1);
    }

    /* Open the FIFO for reading.     */
    if ((fd = open(fifoname, O_RDONLY)) < 0) {
        perror("open");
        exit(1);
    }
    for (n=0; n < N; n++) {
        read(fd, &data, sizeof(data));
        sum+=data;
    }
    printf("Sum of received data: %i\n", sum);
    close(fd);
```

```
    unlink(fifoname);
    exit(0);
}

/*productor.c */
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main(int agvc, char * argv[]) {
    int n, fd;
    char data = 1;
    char *filename=argv[1];
    int N = atoi(argv[2]);

    /* Open the FIFO for writing.  It was
       created by the server. */
    if ((fd = open(filename, O_WRONLY)) < 0) {
        perror("open");
        exit(1);
    }

    /*   write N ones to FIFO.    */
    for (n=0; n < N; n++ )
        write(fd, &data, sizeof(data));
    close(fd);
    exit(0);
}
```

b) (30 puntos) Cree un programa productor y otro consumidor. El proceso consumidor es quien crea una zona de memoria compartida de N bytes. El productor escribe, en memoria compartida, N bytes para el proceso consumidor. Cada char escrito almacena un número 1. Cuando el productor ha llenado con unos la zona compartida, éste "avisa" al proceso consumidor quien comienza a sumarlos, muestra la suma de los datos y termina (debería imprimir N).
La ejecución del proceso Consumidor es:
$ consumidor <Nombre_base> <N>
La ejecución del proceso Productor es:
$ productor <Nombre_base> <N>
Donde Nombre_base es el prefijo usado para la zona de memoria compartida y los semáforos que pueda necesitar.

Consumidor:
1 Genralización de nombres para memoria compartida y semáforos.
2 Crear memoria compartida
2 Dimensionar su tamaño
2 Mapear memoria
2 Crear semáforos
1 wait()
3 lectura
1 post()
1 Remover semáforos
1 remover memoria del sistema
Total: 16

Productor:
1 Genralización de nombres para memoria compartida y semáforos.
2 Apertura se memoria compartida
2 Acceso a memoria compartida
2 Apertura de semáforos
1 Wait()
3 escritura
1 Post()
1 Cerrar semáforos
1 cerrar memoria compartida
Total 14

```
/**
 *    gcc consumer.c -lpthread -lrt
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <errno.h>
#include <semaphore.h>

int main(int argc, char * argv[])  {
  char name[50];            // file name
  int n, N = atoi(argv[2]);             // file size
  char nameEmpty[50];
  char nameFull[50];
  int shm_fd;               // file descriptor, from shm_open()
  char *shm_buffer;         // base address, from mmap()
  int sum=0;

  sprintf(name,"%s_%s","/shm", argv[1]);
  sprintf(nameEmpty,"%s_%s","/EMPTY", argv[1]);
  sprintf(nameFull,"%s_%s","/FULL", argv[1]);

  sem_t * empty_sem, *full_sem;

  /* create the shared memory segment as if it was a file */
  shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
  if (shm_fd == -1) {
    printf("prod: Shared memory failed: %s\n", strerror(errno));
    exit(1);
  }

  /* configure the size of the shared memory segment */
  ftruncate(shm_fd, N);

  /* map the shared memory segment to the address space of the process */
  shm_buffer = mmap(0, N, PROT_READ, MAP_SHARED, shm_fd, 0);
  if (shm_buffer == MAP_FAILED) {
    printf("prod: Map failed: %s\n", strerror(errno));
```

```
    close(shm_fd);
    shm_unlink(name);   /* close and shm_unlink*/
    exit(1);
  }
    /** Create two named semaphores     */
    // first remove the semaphore if it already exists
  if (sem_unlink(nameEmpty) == -1)
    printf("Error removing %s: %s\n", nameEmpty, strerror(errno));
  if (sem_unlink(nameFull) == -1)
    printf("Error removing %s: %s\n", nameEmpty, strerror(errno));

    // create and initialize the semaphore
  if ( (empty_sem = sem_open(nameEmpty, O_CREAT, 0666, 1)) == SEM_FAILED)
    printf("Error creating %s: %s\n", nameEmpty, strerror(errno));
  if ( (full_sem = sem_open(nameFull, O_CREAT, 0666, 0)) == SEM_FAILED)
      printf("Error creating %s: %s\n", nameFull, strerror(errno));

  if (sem_wait(full_sem)!=0)
    printf("Error waiting %s\n",strerror(errno));
  else {
    /**
    * Read from mapped shared memory buffer.
    */
    for (n=0; n < N; n++)
      sum+=shm_buffer[n];
    printf("Sum of shared data is %i.\n", sum);
    if (sem_post(empty_sem)!=0)
      printf("Error posting %s\n",strerror(errno));
   }
   sem_close(empty_sem);
   sem_close(full_sem);
   sem_unlink(nameFull);
   sem_unlink(nameEmpty);

  /* remove the mapped memory segment from the address space of the process */
  if (munmap(shm_buffer, N) == -1) {
    printf("prod: Unmap failed: %s\n", strerror(errno));
    exit(1);
  }

  /* close the shared memory segment as if it was a file */
  if (close(shm_fd) == -1) {
    printf("prod: Close failed: %s\n", strerror(errno));
    exit(1);
  }
  return 0;
}

/**
 *    gcc producer.c -lpthread -lrt
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <errno.h>
```

```
#include <string.h>
#include <semaphore.h>

int main(int argc, char * argv[]) {
  char name[50];            // file name
  int n, N = atoi(argv[2]);             // file size
  char nameEmpty[50];
  char nameFull[50];
  int shm_fd;                // file descriptor, from shm_open()
  char *shm_buffer;        // base address, from mmap()
  sem_t * empty_sem, *full_sem;

  sprintf(name,"%s_%s","/shm", argv[1]);
  sprintf(nameEmpty,"%s_%s","/EMPTY", argv[1]);
  sprintf(nameFull,"%s_%s","/FULL", argv[1]);

  /* open the shared memory segment as if it was a file */
  shm_fd = shm_open(name, O_RDWR, 0666);
  if (shm_fd == -1) {
    printf("cons: Shared memory failed: %s\n", strerror(errno));
    exit(1);
  }

  /* map the shared memory segment to the address space of the process */
  shm_buffer = mmap(0, N, PROT_WRITE, MAP_SHARED, shm_fd, 0);
  if (shm_buffer == MAP_FAILED) {
    printf("cons: Map failed: %s\n", strerror(errno));
    close(shm_fd);
    exit(1);
  }

    /**   Open the two named semaphores     */
    // create and initialize the semaphore
  if ( (empty_sem = sem_open(nameEmpty, 0)) == SEM_FAILED)
      printf("Error opening %s: %s\n",nameEmpty, strerror(errno));
  if ( (full_sem = sem_open(nameFull, 0)) == SEM_FAILED)
      printf("Error opening %s: %s\n",nameFull, strerror(errno));

  if (sem_wait(empty_sem)!=0)
      printf("Error waiting %s\n",strerror(errno));
  else {
      for (n=0; n<N; n++)
        shm_buffer[n]=1;
      if (sem_post(full_sem)!=0)
         printf("Error posting %s\n",strerror(errno));
  }
  sem_close(empty_sem);
  sem_close(full_sem);

  /* remove the mapped shared memory segment from the address space of the process */
  if (munmap(shm_buffer, N) == -1) {
    printf("cons: Unmap failed: %s\n", strerror(errno));
    exit(1);
  }
  /* close the shared memory segment as if it was a file */
  if (close(shm_fd) == -1) {
    printf("cons: Close failed: %s\n", strerror(errno));
    exit(1);
  }
```

```
  return 0;
}
```

c) (5 puntos) Muestre el comando que le permite medir el tiempo de ejecución del programa productor en ambos casos.

```
$ time productor <Nombre_base> <N>
```