Primer Certamen
Tiempo 120 min. Al terminar, entregue sus respuestas vía AULA.

1.- (30 pts) El comando last de Linux muestra la lista de los usuarios que se conectaron a la máquina desde una fecha en adelante. Cree el script bash mfu.sh (most frequent users), el cual lista el nombre de usuario completo (Ej. agustin.gonzalez) mostrados por last y el número de conexiones hechas por cada uno de ellos según el listado de last. La salida muestra desde el nombre de usuario con más conexiones en el periodo al de menos conexiones.
Un ejemplo de su ejecución podría arrojar (si last mostrara sólo 3 usuarios con múltiples conexiones):
$ ./mfu.sh
fernando.cataldov 3
felipe.tapia 2
agustin.gonzalez 1

Ayuda: considere last -w

R: Puede descargar el código desde aquí.

```
#! /bin/bash
last -w | head -n -2 | while read user rest
do
  echo $user
done  | sort | uniq -c| sort -r | while read freq user
do
  echo -e "$user \t $freq"
done
```

Distribución de puntaje:
6 pts Filtrar nombre de usuarios de last
6 pts Ordenar nombres de usuarios
6 pts. Contar número de conexiones para un usuario
6 pts. Ordenar por número de conexiones
6 pts. Listar en orden, número de usuario Nº de conexiones

2.- (35 pts) Desarrolle el programa en lenguaje C bpsPlot.c, que muestra un gráfico para la tasa en bit/s enviados por la interfaz indicada como parámetro. El gráfico se muestra al final, debe considerar un punto por segundo y debe cubrir un periodo de 60 segundos.
Una ejecución del programa en aragorn podría ser:
$ ./bpsPlot eth0

El programa debe mostrar la gráfica al final usando gnuplot y el comando ifconfig. Desarrolle una solución básica pensando en aragorn.elo.utfsm.cl

Ayudas: considere el comando ifconfig <interfaz> para obtener la información del número de bytes transmitidos por una interfaz.
Use un parámetro para el número de puntos o para el tiempo entre puntos, así será rápido hacer pruebas si lo desea.

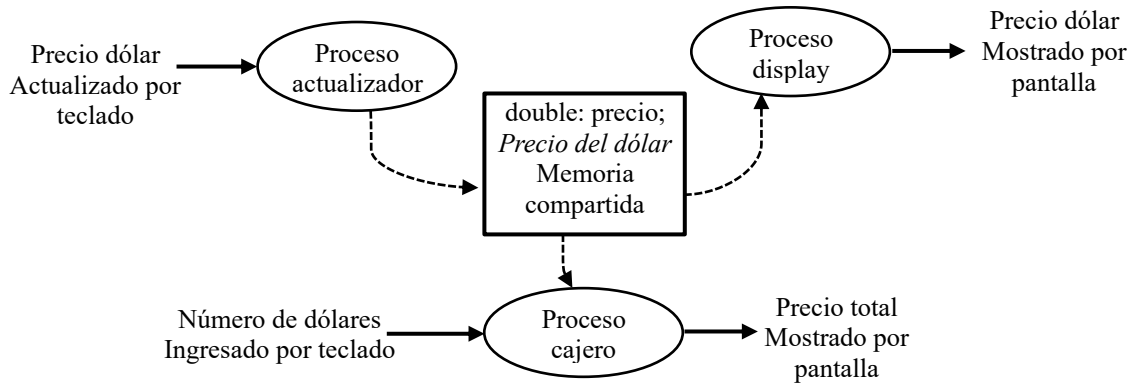
R: Puede descargar el código desde aquí.
Distribución de puntaje
10 pts. Mecanismo para muestreo periódico de datos (6 pts. si usa sleep)

12 pts. Toma de datos usando ifconfig
6 pts. Invocación a ifconfig
6 pts. Filtrar número de bytes transmitidos
6 pts. Almacenamiento de bps
7 pts. Generación del gráfico con gnuplot

```
// to compile:  $gcc -o bpsPlot bpsPlot.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <signal.h>

int numSamples = 60;

static void sig_alarm(int signo) {       /* argument is signal number */
   return;
}

int main(int argc, char* argv[]) {
   int i;
   FILE *pf;
   char cmd[256];
   char junk[64];
   long totalTxBytes_i, totalTXBytes_i_1;

   struct itimerval timerval;
   struct timeval period;
   int signo;
   FILE * outData;

   period.tv_sec=1;
   period.tv_usec=0;
   timerval.it_interval=timerval.it_value=period;
   setitimer(ITIMER_REAL, & timerval, NULL);
   signal(SIGALRM, sig_alarm);

   sprintf(cmd, "ifconfig %s | grep \"TX packets\" ", argv[1]);
   outData = fopen("outData.txt", "w");

   for (i=-1; i <numSamples; i++) {
```

```
        pause();
        pf = popen(cmd, "r");
        fscanf(pf, "%s %s %s %s %ld", junk, junk, junk, junk, &totalTxBytes_i);
        pclose(pf);
        if (i>=0)
            fprintf(outData, "%i %i\n", i,8*(totalTxBytes_i - totalTXBytes_i_1));
        totalTXBytes_i_1 = totalTxBytes_i;
    }
    period.tv_sec=0;
    timerval.it_interval=timerval.it_value=period;
    setitimer(ITIMER_REAL, & timerval, NULL);  //stop timer
    fflush(outData);
    close(outData);
    pf = popen("gnuplot", "w");
    fprintf(pf, "plot \"%s\" using 1:2 with lines lt 1\n", "outData.txt");
    fprintf(pf, "pause -1 \n"); fflush(pf);
    sleep(5);
    fprintf(pf, "\n exit"); fflush(pf);
    pclose(pf);
    exit(0);
}
```

3.- (35 pts) En una oficina de cambio hay tres procesos (**actualizador**, **display** y **cajero**) corriendo. Los tres manipulan el valor en pesos de 1 dólar. **actualizador.c** espera que el usuario ingrese por consola actualizaciones para el valor del dólar. **Actualizador** es el primero en ser ejecutado, al partir crea una variable double (precio) compartida entre los tres procesos y la actualiza cada vez que su usuario ingresa un precio nuevo. **display.c** espera que el valor compartido del dólar (precio) sea actualizado y lo muestra por la pantalla (su salida estándar). **cajero.c** espera por el ingreso por teclado de una cantidad de dólares y usa la variable compartida (precio) para calcular e informar por pantalla el monto en pesos que un cliente debe entregar al cajero para recibir los dólares solicitados.
Desarrolle los programas: **actualizador.c, display.c y cajero.c**.



R: Puede descargar el código desde aquí.
Distribución de puntaje:
15 pts. Actualizador
- 4 pts. Creación de memoria compartida
- 4 pts. Creación e inicialización de semáforos
- 7 pts. Lectura y actualización de precio con buen uso de semáforos

10 pts. Display
- 3 pts. Acceso a memoria compartida
- 3pts. Acceso a semáforos
- 4pts. Lectura y despliegue de precio con buen uso de semáforos

10 pts. Cajero
- 3 pts. Acceso a memoria compartida
- 3pts. Acceso a semáforos
- 4pts. Lectura de cantidad e informe del total con buen uso de semáforo

```c
// to compile: $gcc -o actualizador actualizador.c -lrt -lpthread
#include <stdio.h> // printf
#include <stdlib.h>   // exit
#include <fcntl.h>    //O_CREAT
#include <sys/shm.h>
#include <sys/mman.h>   //mmap
#include <semaphore.h>

int main(void)  {
  const char *nameShm = "/ELO330_C1_2s24_SHM";  /* file name */
  const int SIZE = sizeof(double);    /* file size */
  const char *nameFree = "/ELO330_C1_2s24_FREE";
  const char *nameNew = "/ELO330_C1_2s24_NEW";

  int shm_fd;     /* file descriptor, used with shm_open() */
  double *price;  /* base address, used with mmap()  */
  double newPrice;
  sem_t * free_sem, *new_sem;

  /* create the shared memory segment as if it were a file */
  shm_fd = shm_open(nameShm, O_CREAT | O_RDWR, 0666);
  /* configure the size of the shared memory segment */
  ftruncate(shm_fd, SIZE);
  /* map the shared memory segment to the address space of the process */
  price = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);  // just for writing
  /** Create two named semaphores
  /* first remove the semaphore if each one already exists */
  sem_unlink(nameFree);
```

```
  sem_unlink(nameNew);
  /* create and initialize the semaphore  */
  free_sem = sem_open(nameFree, O_CREAT, 0666, 1);
  new_sem = sem_open(nameNew, O_CREAT, 0666, 0);
  do {
    printf("New price: ");
    scanf("%lf", &newPrice);
    sem_wait(free_sem);
   /** Write to the mapped shared memory region.  */
    *price=newPrice;
    sem_post(new_sem);
    sem_post(free_sem);
  } while (newPrice > 0);
  sem_close(free_sem);
  sem_close(new_sem);
  /* remove the mapped memory segment from the address space of the process */
  munmap(price, SIZE);
  /* close the shared memory segment as if it was a file */
  close(shm_fd);
  exit(0);
}

// to compile: $gcc -o display display.c -lrt -lpthread
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <semaphore.h>

int main(void) {
  const char *nameShm = "/ELO330_C1_2s24_SHM";  /* file name  */
  const int SIZE = sizeof(double);    /* share memory size  */
  const char *nameFree = "/ELO330_C1_2s24_FREE";
  const char *nameNew = "/ELO330_C1_2s24_NEW";

  int shm_fd;     // file descriptor, used with shm_open()
  double *newPrice;  /* base address, used with mmap()  */
  sem_t * free_sem, *new_sem;
```

```
  /* open the shared memory segment as if it were a file */
  shm_fd = shm_open(nameShm, O_RDONLY, 0666);
  /* map the shared memory segment to the address space of the process */
  newPrice = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
  /** Open the two named semaphores   */
  /* create and initialize the semaphore  */
  free_sem = sem_open(nameFree, 0);
  new_sem = sem_open(nameNew, 0);
  do {
    sem_wait(new_sem);
    sem_wait(free_sem);
    /* read from the mapped shared memory segment */
    printf("%.2f\n", *newPrice);
    sem_post(free_sem);
  } while (*newPrice > 0);
  sem_close(free_sem);
  sem_unlink(nameFree);
  sem_close(new_sem);
  sem_unlink(nameNew);
  /* remove the mapped shared memory segment from the address space of the process */
  munmap(newPrice, SIZE);
  /* close the shared memory segment as if it was a file */
  close(shm_fd);
  exit(0);
}

// to compile: $gcc -o cajero cajero.c -lrt -lpthread
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <semaphore.h>

int main(void) {
  const char *name = "/ELO330_C1_2s24_SHM"; /* file name  */
  const int SIZE = sizeof(double);    /* share memory size  */
  const char *nameFree = "/ELO330_C1_2s24_FREE";
```

```
  int shm_fd;     // file descriptor, used with shm_open()
  double *price;  /* base address, used with mmap()  */
  double dollars, total;
  sem_t * free_sem, *new_sem;

  /* open the shared memory segment as if it were a file */
  shm_fd = shm_open(name, O_RDONLY, 0666);
  /* map the shared memory segment to the address space of the process */
  price = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
  /** Open the two named semaphores  */
  /* create and initialize the semaphore  */
  free_sem = sem_open(nameFree, 0);
  do {
    printf("Amount in dollars: ");
    scanf("%lf", &dollars);
    sem_wait(free_sem);
    /* read from the mapped shared memory segment */
    total = dollars*(*price);
    sem_post(free_sem);
    printf("Amount in pesos: %.2f\n", total);
  } while (total > 0);
  sem_close(free_sem);
  sem_unlink(nameFree);
  /* remove the mapped shared memory segment from the address space of the process */
  munmap(price, SIZE);
  /* close the shared memory segment as if it was a file */
  close(shm_fd);
  /* remove the shared memory segment from the file system */
  shm_unlink(name);
  exit(0);
}
```