Primer Certamen
Tiempo 110 min. Al terminar, entregar sus respuestas vía AULA.

1.- (30 puntos) Cree el script shell wdu.sh (por working days of a user) el cual recibe como parámetro el nombre lógico de un/a usuario (Ej. $ ./wdu.sh agustin.gonzalez) y muestra por pantalla los días del mes en que ese/a usuario/a se ha conectado a la máquina donde se correo el script.
El formato de salida debe señalar en nombre "real" del usuario (mismo que muestra finger) y la máquina donde se ejecuta el script
Ejemplo:
agustin.gonzalez@Aragorn:~$ ./wdu.sh agustin.gonzalez
Agustin Gonzalez ha trabajado en Aragorn los siguientes días:
Thu Jan 16
Wed Jan 8
Tue Jan 7
Tue Jan 7
Por ejemplo, su programa debe funcionar en el servidor aragorn.elo.utfsm.cl.
Hint: En su solución considere el comando last con la opción -w, comando finger y hostname.

```
#! /bin/bash
#wdu.sh user
user=$1
finger $1 > /tmp/$$myfile
read login username name realname < /tmp/$$myfile
echo $realname ha trabajado en `hostname` los siguientes días:
last -w | grep $1 | while read user pts ip Day Month Num rest
do
  echo $Day $Month $Num
done
rm /tmp/$$myfile

#2° opción:
set -- `finger $1 | grep Login:`
echo $4 $5 ha trabajado en `hostname` los siguientes días:
last -w | grep $2 | while read user pts ip Day Month Num rest
do
  echo $Day $Month $Num
done

#3° opción:
finger $1 | (read login username name realname; echo -n $realname)
echo " ha" trabajado en `hostname` los siguientes días:
last -w | grep $1 | while read user pts ip Day Month Num rest
do
  echo $Day $Month $Num
done

#4° opción:
finger $1 | (read login username name realname
echo $realname ha trabajado en `hostname` los siguientes días:)
```

```
last -w | grep $1 | while read user pts ip Day Month Num rest
do
  echo $Day $Month $Num
done
```

2.- (30 puntos) Cree el programa graficar.c, en lenguaje C, que pida ingresar 10 números enteros por consola y luego haga una gráfica con esos 10 valores.
Ejemplo de ejecución:
$ ./graficar
Ingrese 10 valores enteros: 23 24 25 26 28 30 30 29 27 25
/* aquí el programa muestra una gráfica con una curva que pasa por las coordenadas (1,23) (2,24) (3,25) (4,26) (5,28) etc. luego de 5 segundos se termina el programa. */
Hint: Considere el uso de gnuplot para hacer la gráfica.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(void) {
   pid_t pid;
   int pfd[2];
   int i, status, data;
   FILE * sd, *tmp;
   char line[256];
   /* Create a pipe. */
   if (pipe(pfd) < 0) {
      perror("pipe");
      exit(1);
   }
   /* Create a child process. */
   if ((pid = fork()) < 0) {
      perror("fork");
      exit(1);
   }
   /* The child process executes "gnuplot". */
   if (pid == 0) {
      /* Attach standard input of this child process to read from the pipe. */
      dup2(pfd[0], 0);
      close(pfd[1]);  /* close the write end off the pipe */
      execlp("gnuplot", "gnuplot", NULL);
      perror("exec");
      _exit(127);
   }
   /* We won't be reading from the pipe. */
   close(pfd[0]);
   sd = fdopen(pfd[1], "w");
   tmp = fopen("/tmp/mytmpFile.txt", "w");
   printf("Ingrese 10 valores enteros:");
   for (i=1; i < 11; i++) {
     scanf("%i", &data);
     fprintf(tmp,"%i  %i \n", i, data);
   }
   fclose(tmp);
   fprintf(sd, "plot \"%s\" with lines lt 1 \n", "/tmp/mytmpFile.txt");
   fflush(sd);
   sleep(5);

   fprintf(sd, "\n exit"); fflush(sd);
```

```
    /* Close the pipe and wait for the child to exit. */
    fclose(sd);
    waitpid(pid, &status, 0);
    exit(0);
}
```

3.- (40 puntos) Se pide hacer un programa (sensor.c) que simula un sensor de temperatura ("productor") el cual cada un segundo actualiza con un valor aleatorio una variable entera que se encuentra en memoria compartida. Cree también un programa lector.c ("consumidor") el cual toma cada valor nuevo generado por el sensor y lo muestra por pantalla. El programa sensor.c termina cuando ha generado X valores, siendo X un argumento del programa sensor.c. El programa lector.c termina luego de mostrar el último valor de temperatura.
Ayuda: revise el ejemplo: http://profesores.elo.utfsm.cl/~agv/elo330/programs/ipc/POSIX_shm/concurrentProducerConsumer/

```
/**
 *    gcc -o sensor sensor.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[])  {
  const char *name = "/C1shm-example"; // file name
  const int SIZE = sizeof(int);     // shared memory size
  const char *nameEmpty = "/C1EMPTY";
  const char *nameFull = "/C1FULL";
  int shm_fd;     // file descriptor, from shm_open()
  int *temperature;  // base address, from mmap()
  sem_t * empty_sem, *full_sem;
  int X = atoi(argv[1]);

  /* 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, SIZE);
  /* map the shared memory segment to the address space of the process */
  temperature = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
  if (temperature == 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", nameFull, 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));

    for (int i=0; i <= X; i++) {
      if (sem_wait(empty_sem)!=0)
          printf("Error waiting %s\n",strerror(errno));
      else {
          /**
           * Write to the mapped shared memory region.
           */
           *temperature = i!=X ? random(): -1;
           if (sem_post(full_sem)!=0)
             printf("Error posting %s\n",strerror(errno));
        sleep(1);
       }
    }
    sem_close(empty_sem);
    sem_close(full_sem);
   /* remove the mapped memory segment from the address space of the process */
   if (munmap(temperature, SIZE) == -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 -o lector lector.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(void) {
   const char *name = "/C1shm-example"; // file name
   const int SIZE = 4096;      // file size
   const char *nameEmpty = "C1EMPTY";
   const char *nameFull = "C1FULL";
   int shm_fd;     // file descriptor, from shm_open()
   int *temperature;  // base address, from mmap()
   sem_t * empty_sem, *full_sem;
```

```
  /* open the shared memory segment as if it was a file */
  shm_fd = shm_open(name, O_RDONLY, 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 */
  temperature = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
  if (temperature == MAP_FAILED) {
    printf("cons: Map failed: %s\n", strerror(errno));
    // close and unlink?
    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));
    do {
      if (sem_wait(full_sem)!=0)
          printf("Error waiting %s\n",strerror(errno));
      else {
          /* read from the mapped shared memory segment */
          if(*temperature > 0) printf("%i\n", *temperature);
          if (sem_post(empty_sem)!=0)
            printf("Error posting %s\n",strerror(errno));
      }
    } while (*temperature > 0);
    sem_close(empty_sem);
    sem_unlink(nameEmpty);
    sem_close(full_sem);
    sem_unlink(nameFull);
  /* remove the mapped shared memory segment from the address space of the process */
  if (munmap(temperature, SIZE) == -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);
  }
  /* remove the shared memory segment from the file system */
  if (shm_unlink(name) == -1) {
    printf("cons: Error removing %s: %s\n", name, strerror(errno));
    exit(1);
  }
  return 0;
}
```