Primer Certamen: Tiempo: 11:45 hrs - 13:30 hrs.
Todas las preguntas tienen igual puntaje.

1.- En sistemas Linux podemos borrar todos los archivos *.o y *.class bajo un directorio (ej. /home/agustin/tareas/) con la instrucción:
$ rm -f `find /home/agustin/tareas/ -name *.class *.o`
Sin embargo cuando los archivos a borrar contienen espacios en su nombre como en "Tarea 2009.o" el comando previo falla.

Cree el script shell **borrar.sh** que consiga el propósito buscado incluso con archivos con espacios en su nombre. borrar.sh acepta como argumento el directorio bajo el cual se borrarán los archivos señalados. Un ejemplo de ejecución es:
$ borrar.sh /home/agustin/tareas/

```
#! /bin/bash
find $1 -name *.class | while read line
do
  rm -f "$line"
done
find $1 -name *.o | while read line
do
  rm -f "$line"
done
```

Otra solución más compacta:

```
#! /bin/bash
find $1 -name *.class -o -name *.o | while read line
do
  rm -f "$line"
done
```

Otra, más compacta aún (basada en respuesta de Jorge Pareja y Daniel Velásquez)

```
#! /bin/bash
find $1 \( -name *.class -o -name *.o \) -exec rm -f {} \;
```

Nota: Inicialmente la pregunta borraba *.o; luego la extendí a *.o y *.class, pero no me dí cuenta del cambio que esto implica en la sitaxis en el comando find.

2.- Haga un programa que espere por la llegada de Control-C, luego duerma por dos segundos, imprima por pantalla el mensaje "Llegó Control-C.\n" y termine. El programa debe ignorar otras señales como SIGUSR1.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

int done=0;
static void sig_usr(int signo)  {
  done = 1;
  return;
}

int main(void)  {
  if (signal(SIGINT, sig_usr) == SIG_ERR)
    exit(-1);

  while (!done)
    pause();
  sleep(2);
  printf("Llegó Control-C\n");
  exit(0);
}
```

Nota: Soluciones en que el programa termina en la atención de la señal fueron aceptadas; sin embargo, la solución aquí sugerida es más general. El programa puede continuar con cualquier cosa (en lugar del sleep) después de la llegada del Control-C.

3.- El utilitario bc permite calcular expresiones aritméticas variadas. Por su entrada estándar recibe la expresión a calcular y por la salida estándar envía el resultado. Para terminar su operación debemos ingresar quit. Un ejemplo de su uso donde pedidos evaluar dos expresiones es:

```
$ echo -e "3+4\n   2.4*(3-8.9)\n  quit" | bc
7
-14.1
$
```

Cree el programa **ea,** evaluador aritmético, programa en C que repetidamente pide ingresar por teclado una expresión aritmética y retorna su valor. Para terminar el usuario debe ingresar quit.

Nota: La idea es que a futuro usted sepa cómo incluir un evaluador aritmético en sus programas. bc puede hacer mucho más.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void) {
  pid_t pid;
  int pfdw[2];  /* pipe to send data form parent to child */
  int pfdr[2];  /* pipe to read data form child */
  int nc, status;
  char expresion[100];
  char respuesta[100];

  if (pipe(pfdw) < 0) {
    perror("pipe");
    exit(1);
  }
  if (pipe(pfdr) < 0) {
    perror("pipe");
    exit(1);
  }
  if ((pid = fork()) < 0) {
    perror("fork");
    exit(1);
  }
  if (pid == 0) {  /* This is the child */
      /* Attach standard input to the end of the pipe where parent write.     */
    dup2(pfdw[0], 0);
    close(pfdw[1]);
      /* Attach standard output to the end of the pipe where parent read.    */
    dup2(pfdr[1],1);
    close(pfdr[0]);
    execlp("bc", "bc", 0);
    perror("exec");
    _exit(127);
```

```
  }
  close(pfdw[0]);
  close(pfdr[1]);

  do
  {
    nc=read(STDIN_FILENO, expresion, sizeof(expresion));
    expresion[nc]='\n';
    write(pfdw[1], expresion, nc+1);
    nc=read(pfdr[0], respuesta, sizeof(respuesta));
     write(STDOUT_FILENO, respuesta, nc);
  } while (strncmp(expresion, "quit", 4)!=0);
    /* Close the pipe and wait for the child to exit.  */
  close(pfdw[1]);
  close(pfdr[0]);
  waitpid(pid, &status, 0);
  exit(0);
}
```

Otra solución aceptable a la pregunta con popen. Esta solución es menos general pues el programa padre puede desear hacer algo con el reultado y no necesariamente enviarlo a pantalla como en este caso.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE * bc;
  int nc;
  char expresion[100];

  if ((bc = popen("bc", "w")) == NULL) {
    perror("popen");
    exit(1);
  }
  do  {
    nc=read(STDIN_FILENO, expresion, sizeof(expresion));
    expresion[nc]='\n';
    fprintf(bc, "%s\n", expresion); fflush(bc);
  } while (strncmp(expresion, "quit", 4)!=0);
  pclose(bc);
  exit(0);
}
```

4.- Cree el programa testHilos.c. Éste acepta como argumento 0 ó 1. Lo componen dos hilos que acceden una misma variable entera inicialmente en cero. Una hebra incrementa ese entero 2.000.000 veces y retorna el valor máximo alcanzado por la variable. La otra decrementa 2.000.000 veces el entero, calcula el valor mínimo alcanzado por éste, espera el término de la hebra de incremento, muestra por pantalla el valor retornado ésta y el valor final de la variable entera. Con argumento 1 el

acceso a la variable es controlado para evitar inconsistencias. Con argumento 0, los accesos son sin control de exclusión.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define ITERACIONES 2000000

int contador;
int max;
int exclusion;
pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;

void * increment(void * arg)
{
  int i;
  max=contador;
  for (i=0; i<ITERACIONES; i++){
    if (exclusion)
      pthread_mutex_lock(&mylock);
    contador++;
    if (contador > max)
      max=contador;
    if (exclusion)
      pthread_mutex_unlock(&mylock);
  }
  return (&max);
}

int main(int argc, char * argv[])
{
  int err, i;
  pthread_t tid;
  int min=contador;
  int status;

  exclusion = atoi(argv[1]);
  contador = 0;
  err = pthread_create(&tid, NULL, increment, NULL);
  if (err != 0){
    printf("can't create thread \n");
    exit(0);
  }
  for (i = 0; i <ITERACIONES; i++){
    if (exclusion)
      pthread_mutex_lock(&mylock);
    contador--;
    if (contador < min) min = contador;
    if (exclusion)
```

```
    pthread_mutex_unlock(&mylock);
  }
  pthread_join(tid, (void*)&status);
  printf("El valor mínimo de contador es %i\n", min);
  printf("El valor máximo de contador es %i\n", status);
  printf("El valor final de contador es %i\n", contador);
}
```