PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
Examen 1
(Primer semestre de 2019)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. A. Bello R.

Duración:	3 horas
Nota:	No se puede usar ningún material de consulta. **La presentación, la ortografía y la gramática influirán en la calificación.** **El examen debe ser desarrollado en el cuadernillo usando lapicero.** **Lo escrito con lápiz será considerado como borrador y NO será evaluado.**
Puntaje total:	20 puntos

Pregunta 1 (2 puntos – 16 min.)
a) (1 punto – 8 min.) "Although third-generation operating systems were well suited for big scientific calculations and massive commercial data-processing runs, they were still basically batch systems. Many programmers pined for the first-generation days when they had the machine all to themselves for a few hours, so they could debug their programs quickly. With third-generation systems, the time between submitting a job and getting back the output was often several hours, so a single misplaced comma could cause a compilation to fail, and the programmer to waste half a day. Programmers did not like that very much. This desire for quick response time paved the way for ..., a variant of multiprogramming, in which each user has an online terminal." What is a variant of multiprogramming we are talking about?

b) (1 punto – 8 min.) "Files in UNIX are protected by assigning each one a 9-bit binary protection code. The protection code consists of three 3-bit fields, one for the owner, one for other members of the owner's group (users are divided into groups by the system administrator), and one for everyone else. Each field has a bit for read access, a bit for write access, and a bit for execute access. These 3 bits are known as the **rwx bits**. For example, the protection code *rwxr-x--x* means that the owner can **r**ead, **w**rite, or e**x**ecute the file, other group members can read or execute (but not write) the file, and everyone else can execute (but not read or write) the file. For a directory, *x* indicates … . A dash means that the corresponding permission is absent." What does *x* indicate for a directory?

Pregunta 2 (4 puntos – 32 min.) Sin explicar el código del programa, explique qué pretende realizar este programa y de qué manera.

```
$ cat -n 2019-1_ex1.c
     1	#include <stdio.h>
     2	#include <stdlib.h>
     3	#include <unistd.h>
     4	#include <string.h>
     5	#include <signal.h>
     6	
     7	void sig_usr_handler(int signum);
     8	void die(char *s);
     9	
    10	char *path;
    11	char *progname;
    12	char *ps = "ps | wc -l"; /* cuenta cantidad de líneas en salida de ps */
    13	int flag = 0;
    14	
    15	int
    16	main(int argc, char **argv) {
    17	    pid_t father, pid, ppid;
    18	    char arg1[8];
    19	    FILE *s2f;
    20	    char buf[10];
    21	    int n0, i, n;
    22	    int pidpipe[2];
    23	
    24	    path = argv[0];
    25	    progname = ((progname=strrchr(argv[0],'/'))? ++progname : argv[0]);
    26	    /* char *strrchr(const char *s, int c);
    27	       The strrchr() function returns a pointer to the last occurrence of  the
    28	       character c in the string s. */
    29	    if (signal(SIGUSR1,sig_usr_handler) == SIG_ERR) die("signal() error\n");
```

```c

    if (argc == 2) {
        pid = getpid();
        write(1,&pid,sizeof pid); dup2(1,pidpipe[1]); close(1);
        sleep(5);
        if (!flag) {
            for (i=0; i<2; i++) {
                if ((pid=fork()) == -1) die("fork() error\n");
                if (!pid) {
                    dup2(pidpipe[1],1);
                    execl(path,progname,argv[1],NULL);
                    die("execl() error\n");
                } /* if !pid */
            } /* for */
        } /* if !flag */
        for (;;) {
            close(pidpipe[1]);
            if (flag) {
                sscanf(argv[1],"%d",&father);
                ppid = getppid();
                if (father != ppid && ppid != 1) kill(ppid,SIGKILL);
                flag = 0;
            }
        } /* for */
        pause; /* wait for signal */
        die(NULL);
    }

    father = getpid();
    sprintf(arg1,"%d",father);

    if (!(s2f=popen(ps,"r"))) die("popen() error\n");
    /* FILE *popen(const char *command, const char *type);
       The  popen()  function opens a process by creating a pipe, forking, and
       invoking the shell.  Since a pipe is by definition unidirectional,  the
       type  argument  may  specify  only  reading  or  writing, not both; the
       resulting stream is correspondingly read-only or write-only. */
    fgets(buf,sizeof buf,s2f);
    pclose(s2f);
    sscanf(buf,"%d",&n0);

    if (pipe(pidpipe) < 0) die("pipe() error\n");
    dup2(pidpipe[0],0); dup2(pidpipe[1],1);
    close(pidpipe[0]); close(pidpipe[1]);
    for (i=0; i<2; i++) {
        if ((pid=fork()) == -1) die("fork() error\n");
        if (!pid) {
            execl(path, progname, arg1, NULL);
            die("execl() error\n");
        }
    }
    close(1);
    while(1) {
        sleep(1);
        if (!(s2f=popen(ps,"r"))) die("popen() error\n");
        fgets(buf,sizeof buf,s2f);
        pclose(s2f);
        sscanf(buf,"%d",&n);
        fprintf(stderr,"n = %d\n",n);
        if (flag) {
            fprintf(stderr, "Misión cumplida.\n");
            die(NULL);
        }
        if ((n-n0)>5) {
            while (read(0,&pid,sizeof pid)) kill(pid,SIGUSR1);
            kill(getpid(),SIGUSR1);
        }
    }
}

void
sig_usr_handler(int signum) {
    flag = 1;
    return;
}

void
die(char *s) {
    if (s != (char *)NULL) {
        while (*s) (void) write(2,s++,1);
    }
    exit((s == (char *)NULL)? 0 : 1);
}
```

```
$ gcc -o 2019-1_ex1 2019-1_ex1.c
$ ./2019-1_ex1 &
[1] 22958
$ ...
```

Pregunta 3 (4 puntos – 32 min.) While humans can see and hear each other, computers can only read and write. So, one computer can write a note (or send a message) that the other computer will later read, but they cannot see each other. To understand the difficulty with this type of restricted communication, let us examine a simple two-person interactions where communication is restricted to writing and reading of notes. The two people involved, let's call them Alice and Bob, can not see each other and they communicate only by writing and reading of notes. In particular, Alice can not see that Bob is reading a note that she has written to him earlier. Alice and Bob are sharing an apartment. Alice arrives home in the afternoon, looks in the fridge and finds that there is no milk. So, she leaves for the grocery to buy milk. After she leaves, Bob arrives, he also finds that there is no milk and goes to buy milk. At the end they both buy milk and end up with too much milk. So, Alice and Bob are looking for a solution to ensure that: (1) only one person buys milk, when there is no milk; (2) someone always buys milk, when there is no milk.

For this solution, four labelled notes are used. Alice uses notes *A1* and *A2*, while Bob uses notes *B1* and *B2*. At any point, if Alice (Bob) finds that there is no note labelled *B1* (*A1*) on the fridge's door, then it is Alice (Bob) responsibility to buy milk. Otherwise, when both *A1* and *B1* are present, a decision is made according to the notes *A2* and *B2*. If both *A2* and *B2* are present or if neither of them is present than it is Bob's responsibility to buy milk, otherwise it is Alice's responsibility:

PROGRAM FOR ALICE:

```
1    leave note A1
2    if B2
3         then leave note A2
4         else remove note A2 fi
5    while B1 and
6       ((A2 and B2) or
7        (no A2 and no B2))
8         do skip od
9    if (no milk) then buy milk fi
10   remove note A1
```

PROGRAM FOR BOB:

```
1    leave note B1
2    if (no A2)
3         then leave note B2
4         else remove note B2 fi
5    while A1 and
6       (( ... ) or
7        ( ... ))
8         do skip od
9    if (no milk) then buy milk fi
10   remove note B1
```

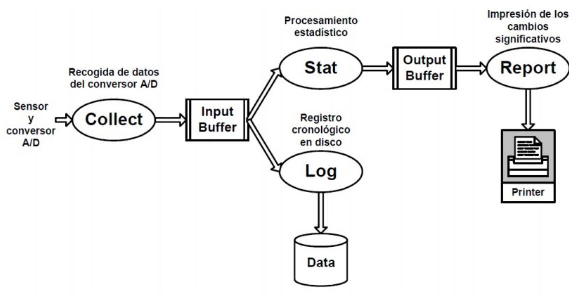
Complete la solución.

Pregunta 4 (4 puntos – 32 min.) Interprocess communications

a) (2 puntos – 16 min.) El siguiente código en C, implementa barreras empleando variables de condición. Complete el código para que el programa funcione.

```c
#include <errno.h>
#include <pthread.h>

static pthread_cond_t bcond = PTHREAD_COND_INITIALIZER;
static pthread_mutex_t bmutex = PTHREAD_MUTEX_INITIALIZER;
static int count = 0;
static int limit = 0;

int initbarrier(int n) {              /* initialize the barrier to be size n */
   int error;

   if (error = pthread_mutex_lock(&bmutex))        /* couldn't lock, give up */
      return error;
   if (limit != 0) {                 /* barrier can only be initialized once */
      pthread_mutex_unlock(&bmutex);
      return EINVAL;
   }
   limit = n;
   return pthread_mutex_unlock(&bmutex);
}

int waitbarrier(void) {    /* wait at the barrier until all n threads arrive */
   int berror = 0;
   int error;

   if (error = pthread_mutex_lock(&bmutex))        /* couldn't lock, give up */
      return error;
   if (limit <=  0) {                        /* make sure barrier initialized */
      pthread_mutex_unlock(&bmutex);
      return EINVAL;
   }
   count++;
   . . .                  /* <------ complete aquí */
   if (!berror)
      berror = pthread_cond_broadcast(&bcond);          /* wake up everyone */
   error = pthread_mutex_unlock(&bmutex);
   if (berror)
      return berror;
   return error;
}
```

b) (2 puntos – 16 min) Se desea simular en Python la sincronización de hilos en le caso del sistema de adquisición de datos, propuesto por Millan Milenkovic, compuesto por los siguientes hilos: Collect, Log, Stat y Report. El hilo Collect adquiere los datos del DAC y los deposita en un *buffer* de entrada, el cual es leído por los procesos Stat y Log. A su vez Stat realiza los cálculos y deposita el resultado en *buffer* de salida, el cual es leído por el proceso Report.



Sensor y conversor A/D
Recogida de datos del conversor A/D
Collect
Input Buffer
Procesamiento estadístico
Stat
Output Buffer
Impresión de los cambios significativos
Report
Printer
Registro cronológico en disco
Log
Data

M. Milenkovic, OSCD

A continuación el programa en Python carente de sincronización:

```
import threading

def Collect():
    while True:
        print('C',end=' ')

def Log():
    while True:
        print('L',end=' ')

def Stat():
    while True:
        print('S',end=' ')

def Report():
    while True:
        print('R',end=' ')

if __name__ == "__main__":

       h1=threading.Thread(target=Collect)
       h2=threading.Thread(target=Log)
       h3=threading.Thread(target=Stat)
       h4=threading.Thread(target=Report)
       h1.start()
       h2.start()
       h3.start()
       h4.start()
```

Empleando barreras en Python, sincronice los hilos según el enunciado. Recuerde que para crear una barrera en Python lo tiene que hacer de la siguiente forma: `b = threading.Barrier(n)` donde `n` es el número de hilos que se desea esperar en la barrera. Y para usar la barrera: `b.wait()`

Pregunta 5 (4 puntos – 32 min.) El problema de la cena de los filósofos. La siguiente solución al problema de los filósofos empleando monitores es incorrecta. Muestre un caso por la que falla y proponga una solución.

```
MONITOR CINCO_FILOSOFOS
Type
 tipo_estado = (meditando , esperando_comer , comiendo);
Var
 estado: array [0..4] of tipo_estado;
 espera: array [0..4] of condition;
 cont: integer;
```

```
Procedure INTENTAR_COMER (j : integer)
begin
 if (estado[(j+1) mod 5] = comiendo) or (estado[(j-1) mod 5] = comiendo)
    then wait(espera[j]);
    else estado[j] := comiendo;
end;

Procedure DEJAR_COMER (j : integer)
begin
   estado[j] := meditando;
   if (estado[(j+1) mod 5] = esperando_comer) and (estado[(j+2) mod 5] < > comiendo)
     then begin
             estado[(j+1) mod 5)] := comiendo;
             send (espera[(j+1) mod 5]);
          end;
   if (estado[(j-1) mod 5] = esperando_comer) and (estado[(j-2) mod 5] < > comiendo)
     then begin
            estado[(j-1) mod 5)] := comiendo;
            send (espera[(j-1) mod 5]);
          end;
end;

Begin { monitor }
 for cont:= 0 to 4 do
     estado[cont] := esperando_comer;
End;

-----------------------------------------------------------------------------------

Procedure filosofo (f : integer)
begin
 repeat
   meditar;
   INTENTAR_COMER (f);
   comer;
   DEJAR_COMER (f);
 until false
end;

BEGIN { cuerpo del programa principal }
 Cobegin
   filosofo(0);
   filosofo(1);
   filosofo(2);
   filosofo(3);
   filosofo(4);
 Coend;
END.
```

<u>**Pregunta 6**</u> **(2 puntos – 16 min.)** Planificación de procesos
a) (1 punto – 8 min.) A continuación se muestran las ráfagas de CPU y de E/S de un proceso:

CPU (2 udt)	**E/S (3 udt)**	**CPU (3 udt)**	**E/S (2 udt)**	**CPU (2 udt)**

Se sabe que su tiempo de espera ha sido de 8 udt. Y que el tiempo total necesario para su ejecución ha sido de 25 udt., ¿cuánto tiempo ha estado dicho proceso bloqueado esperando para realizar algunas de las operaciones de E/S?

b) (1 punto – 8 min.) Suponga que en el instante 0 existen dos procesos, cada uno de ellos consta de una sola ráfaga de CPU de 10 udt., ¿cuál es el tiempo de espera medio si el algoritmo de planificación es RR (*Round Robin*) con *quantum* 6 udt.?

Preparado por AB (4-6) y VK (1-3)
con LibreOffice Writer en Linux Mint 19.1 Tessa

Profesores del curso: (0781) V. Khlebnikov
(0782) A. Bello R.

Pando, 15 de mayo de 2019