PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Segundo semestre de 2023)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **Sin apuntes de clase, sin libros y sin calculadora o computadora.**
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (2 puntos – 10 min.)

(a) (0,5 puntos) "Most **...** have two modes of operation: kernel mode and user mode." ¿Qué palabra falta? Responda con una sola palabra.

(b) (0,5 puntos) ¿En qué modo se ejecuta el siguiente programa?

```
PASSWD(1)                       User Commands                       PASSWD(1)

NAME
       passwd - change user password

SYNOPSIS
       passwd [options] [LOGIN]

DESCRIPTION
       The passwd command changes passwords for user accounts. A normal user
       may only change the password for their own account, while the superuser
       may change the password for any account.  passwd also changes the
       account or associated password validity period.
```

(c) (1 punto)

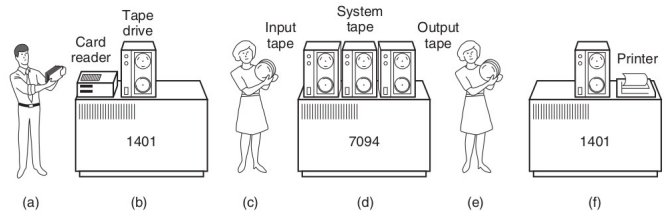

Figure 1-3. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

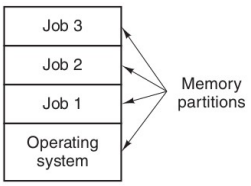

Figure 1-5. A multiprogramming system with three jobs in memory.

¿Por qué el sistema operativo OS/360 coloca Job 1, Job 2 y Job 3 en memoria si los tiene en las cintas magnéticas (como en la computadora IBM 7094)? ¿Cuál es la ventaja de la multiprogramación si las computadoras tenían una sola unidad de procesamiento que no puede procesar 3 (o 4) instrucciones en el mismo instante sino solamente una?

Pregunta 2 (4 puntos – 20 min.)

(a) (1 punto) "… a windowing system called the **X Window System** (also known as **X11**) [was] produced at **…** .". Complete la sentencia.

(b) (1 punto) "… For phone manufacturers, Android, a Linux-based operating system released by Google in 2008, had the advantage that it was **…** and available under a permissive license. As a result, they could tinker with it and adapt it to their own hardware with ease." Complete la sentencia.

(c) (1 punto) "… A CPU might have separate fetch, decode, and execute units, so that while it is executing instruction *n*, it could also be decoding instruction *n* + 1 and fetching instruction *n* + 2. Such an organization is called a **…** ."

(d) (1 punto) "… The difference between the L1 and L2 caches lies in the timing. Access to the L1 cache is done **…** , whereas access to the L2 cache involves a delay of **…** ." ¿Cuáles son los tiempos de acceso a estos dos tipos de memoria?

Pregunta 3 (10 puntos – 50 min.)

```
$ man tee
TEE(1)                          User Commands                          TEE(1)

NAME
       tee - read from standard input and write to standard output and files

SYNOPSIS
       tee [OPTION]... [FILE]...

DESCRIPTION
       Copy standard input to each FILE, and also to standard output.
...

$ which {date,tee,cat}
/usr/bin/date
/usr/bin/tee
/usr/bin/cat

$ ls -l $(which {date,tee,cat})
-rwxr-xr-x 1 root root  35280 feb  7  2022 /usr/bin/cat
-rwxr-xr-x 1 root root 104960 feb  7  2022 /usr/bin/date
-rwxr-xr-x 1 root root  35328 feb  7  2022 /usr/bin/tee

$ date | tee out.log | cat
jue 31 ago 2023 00:51:04 -05

$ date
jue 31 ago 2023 00:51:11 -05

$ cat out.log
jue 31 ago 2023 00:51:04 -05
```

(a) (5 puntos) Aquí usamos tres programas cuyos funcionamientos usted puede imaginar. Pero ¿cuál será el funcionamiento del *shell* ejecutando la orden `$ date | tee out.log | cat` ? Explique en términos de procesos (secuencia de creación de ¿hijos?, ¿nietos?, etc.) (**2 puntos**), mecanismos de comunicación entre ellos (secuencia de creación, considerando que cada proceso nace con 3 archivos abiertos) (**1 punto**), la razón de terminación de cada proceso y la secuencia de terminación de ellos (**1 punto**). ¿Cuál será el valor del descriptor del archivo **`out.log`** durante la ejecución del programa **`tee`**? (**1 punto**)

Ahora consideremos

```
$ man 2 tee
TEE(2)                     Linux Programmer's Manual                    TEE(2)

NAME
       tee - duplicating pipe content

SYNOPSIS
       #define _GNU_SOURCE         /* See feature_test_macros(7) */
       #include <fcntl.h>

       ssize_t tee(int fd_in, int fd_out, size_t len, unsigned int flags);

DESCRIPTION
       tee()  duplicates  up to len bytes of data from the pipe referred to by
       the file descriptor fd_in to the pipe referred to by the file  descrip-
       tor  fd_out.   It  does  not  consume  the data that is duplicated from
       fd_in; therefore, that data can be copied by a subsequent splice(2).
...
RETURN VALUE
       Upon successful completion, tee() returns the number of bytes that were
       duplicated between the input and output.  A return  value  of  0  means
       that  there  was  no  data  to transfer, and it would not make sense to
       block, because there are no writers connected to the write end  of  the
       pipe referred to by fd_in.

       On error, tee() returns -1 and errno is set to indicate the error.
...
```

```
$ man splice
SPLICE(2)                 Linux Programmer's Manual                SPLICE(2)

NAME
       splice - splice (empalmar) data to/from a pipe

SYNOPSIS
       #define _GNU_SOURCE         /* See feature_test_macros(7) */
       #include <fcntl.h>

       ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
                      loff_t *off_out, size_t len, unsigned int flags);

DESCRIPTION
       splice()  moves  data  between two file descriptors without copying be-
       tween kernel address space and user address space.  It transfers up  to
       len bytes of data from the file descriptor fd_in to the file descriptor
       fd_out, where one of the file descriptors must refer to a pipe.
...

$ cat -n tee_1.c
     1   /*
     2    *  man 2 tee
     3    *
     4    *  EXAMPLES
     5    *      The  example  below  implements  a basic tee(1) program using the tee()
     6    *      system call.  Here is an example of its use:
     7    *
     8    *          $ date |./a.out out.log | cat
     9    *          Tue Oct 28 10:06:00 CET 2014
    10    *          $ cat out.log
    11    *          Tue Oct 28 10:06:00 CET 2014
    12    */
    13
    14   #define _GNU_SOURCE
    15   #include <fcntl.h>
    16   #include <stdio.h>
    17   #include <stdlib.h>
    18   #include <unistd.h>
    19   #include <errno.h>
    20   #include <limits.h>
    21
    22   int
    23   main(int argc, char *argv[])
    24   {
    25       int fd;
    26       int len, slen;
    27
    28       if (argc != 2) {
    29           fprintf(stderr, "Usage: %s <file>\n", argv[0]);
    30           exit(EXIT_FAILURE);
    31       }
    32
    33       fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    34       if (fd == -1) {
    35           perror("open");
    36           exit(EXIT_FAILURE);
    37       }
    38
    39       do {
    40           /*
    41            * tee stdin to stdout.
    42            */
    43           len = tee(STDIN_FILENO, STDOUT_FILENO,
    44                     INT_MAX, SPLICE_F_NONBLOCK);
    45
    46           if (len < 0) {
    47               if (errno == EAGAIN)
    48                   continue;
    49               perror("tee");
    50               exit(EXIT_FAILURE);
    51           } else
    52       ...
    53       ...
    54
    55           /*
    56            * Consume stdin by splicing (empalmando) it to a file.
    57            */
    58           while (len > 0) {
    59               slen = splice(STDIN_FILENO, NULL, fd, NULL,
    60                             len, SPLICE_F_MOVE);
    61               if (slen < 0) {
    62                   perror("splice");
    63                   break;
```

```
    64              }
    65              len -= slen;
    66          }
    67      } while (1);
    68
    69      close(fd);
    70      exit(EXIT_SUCCESS);
    71  }
```

```
$ gcc tee_1.c
$ ls -l a.out
-rwxrwxr-x 1 vk vk 16304 ago 31 00:49 a.out
$ date | ./a.out out.log | cat
jue 31 ago 2023 00:51:55 -05
$ date
jue 31 ago 2023 00:51:59 -05
$ cat out.log
jue 31 ago 2023 00:51:55 -05
```

(b) (**2 puntos**) Complete las líneas 52 y 53 del último código presentado.

Realmente el ejemplo de uso **tee**(2) presentado en el programa **tee_1.c** no corresponde a la funcionalidad principal de **tee**(1) (ignorando sus opciones). Mientras que el programa de MINIX 3 (`minix3/commands/simple/tee.c`) corresponde a esa funcionalidad:

```
$ cat -n tee.c
     1  /* tee - pipe fitting           Author: Paul Polderman */
     2
     3  #include <sys/types.h>
     4  #include <fcntl.h>
     5  #include <signal.h>
     6  #include <unistd.h>
     7  #include <stdlib.h>
     8  #include <minix/minlib.h>
     9
    10  #define  MAXFD    18
    11  #define CHUNK_SIZE 4096
    12
    13  int fd[MAXFD];
    14
    15  _PROTOTYPE(int main, (int argc, char **argv));
    16
    17  int main(argc, argv)
    18  int argc;
    19  char **argv;
    20  {
    21      char iflag = 0, aflag = 0;
    22      char buf[CHUNK_SIZE];
    23      int i, s, n;
    24
    25      argv++;
    26      --argc;
    27      while (argc > 0 && argv[0][0] == '-') {
    28          switch (argv[0][1]) {
    29              case 'i':       /* Interrupt turned off. */
    30                  iflag++;
    31                  break;
    32              case 'a':       /* Append to outputfile(s), instead of
    33                               * overwriting them. */
    34                  aflag++;
    35                  break;
    36              default:
    37                  std_err("Usage: tee [-i] [-a] [files].\n");
    38                  exit(1);
    39          }
    40          argv++;
    41          --argc;
    42      }
    43      fd[0] = 1;
    44      for (s = 1; s < MAXFD && argc > 0; --argc, argv++, s++) {
    45          if (aflag && (fd[s] = open(*argv, O_RDWR)) >= 0) {
    46              lseek(fd[s], 0L, SEEK_END);
    47              continue;
    48          } else {
    49              if ((fd[s] = creat(*argv, 0666)) >= 0) continue;
    50          }
    51          std_err("Cannot open output file: ");
    52          std_err(*argv);
    53          std_err("\n");
    54          exit(2);
    55      }
    56
```

```
    57      if (iflag) signal(SIGINT, SIG_IGN);
    58
    59      while ((n = read(0, buf, CHUNK_SIZE)) > 0) {
    60          for (i = 0; i < s; i++) write(fd[i], buf, n);
    61      }
    62
    63      for (i = 0; i < s; i++) /* Close all fd's */
    64          close(fd[i]);
    65      return(0);
    66  }
```

(c) (2 puntos) ¿Por qué el programa **`tee_1.c`** no corresponde a la funcionalidad de **`tee`**(1)?

(d) (1 punto) ¿Por que en la línea 43 del último código `fd[0]` toma justamente el valor 1 y no el otro?

Pregunta 4 (4 puntos – 20 min.) Considerando el siguiente programa:

```
$ cat foo.c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc,char **argv) {
    int n, bar, i;
    char barstr[3];

    if (argc == 1) {
        printf("Mensaje de error\n");
        exit(EXIT_FAILURE);
    }
    n= atoi(argv[1]);
    if (n <= 1) exit(EXIT_SUCCESS);
    if (argc == 2) bar= 0;
    else bar= atoi(argv[2]);
    if (bar < n) {
        sprintf(barstr,"%d", ++bar);
        for (i=0; i<n; ++i)
            if (!fork()) execl("./foo", "foo", argv[1], barstr, (char *)0);
    }
    for (i=0; i<n; ++i) waitpid(-1,NULL,0);
    printf("pid=%d, ppid=%d, argv[2]=%s\n",getpid(),getppid(),argv[2]);
    exit(EXIT_SUCCESS);
}

$ gcc foo.c -o foo
$ ./foo 3

$ date | wc -l
1
$ cal
    Agosto 2023
do lu ma mi ju vi sá
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

$ cal | wc -l
8
$
$ time ./foo 3 | wc -l
...

real 0m0,017s
user 0m0,031s
sys  0m0,026s


$ time ./foo 4 | wc -l
...

real 0m0,020s
user 0m0,207s
sys  0m0,073s
```

```
$ time ./foo 5 | wc -l
...

real 0m0,129s
user 0m1,889s
sys  0m0,377s

$ time ./foo 6 | wc -l
...

real 0m12,169s
user 0m41,906s
sys  1m7,679s
```

Aquí `wc` cuenta las líneas de la salida producida por el programa, mientras que `time` mide el tiempo de trabajo. Se necesita completar la salida de `wc` en estas ejecuciones presentando el cálculo.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 21.2 “Victoria”

Profesor del curso: V. Khlebnikov

Pando, 31 de agosto de 2023