75.08 Sistemas Operativos

Lab Unix - Parte 3

Nombre y Apellido: Damián Cassinotti
Padrón: 96618
Fecha de Entrega: 13/04/2018

Índice

1. Comandos obligatorios

La tercera parte del lab de Unix consiste en el desarrollo de otros comandos: tee, ls, cp y, opcionalmente, ps. Dichos comandos serán versiones simplificadas de los originales. Por cada comando se dará un breve resumen de lo pedido, así como también se contemplarán las precondiciones tomadas.

1.1. tee0

El comando *tee0* deberá tomar como parámetro un archivo, e imprimir tanto en dicho archivo como por salida estándar todo lo que se obtenga de la entrada estándar. Para esto se utilizaron las syscalls open, read, write y close. En caso de que el archivo no exista, se crea; caso contrario, se sobreescribe.

A continuacón se mostrará el código de los archivos utilizados.

```c
#define _POSIX_C_SOURCE 200809L
// open
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
// read
#include <unistd.h>

#define READ_BYTES 1024
int main(int argc, char* argv[]);
void tee0(char* filepath);
void write0(int fd, char *buffer, int cantidad);
```

tee0.h

```c
#include "tee0.h"

int main(int argc, char* argv[]) {
    if (argc != 2)
        return -1;
    tee0(argv[1]);
    return 0;
}

void tee0(char* filepath) {
    int leidos = 0;
    int output_fd = creat(filepath,
    ↪ S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH);
    char buffer[READ_BYTES];
    while ((leidos = read(STDIN_FILENO, buffer, READ_BYTES)) > 0) {
        write0(STDOUT_FILENO, buffer, leidos);
        write0(output_fd, buffer, leidos);
```

```
17      }
18      close(output_fd);
19  }
20
21  void write0(int fd, char *buffer, int cantidad) {
22      int escritos = 0;
23      while (escritos < cantidad) {
24          escritos += write(fd, buffer, cantidad);
25      }
26  }
```

tee0.c

1.2. ls0

El comando *ls0* será una implementación equivalente a *ls –format=single-column –sort=none*, es decir, se listarán todos los archivos del directorio de a uno por linea y sin un orden determinado. La solución consiste en recorrer cada entrada del directorio sobre el cual se está ejecutando el proceso.

La solución propuesta es la siguiente:

```
1 #define _POSIX_C_SOURCE 200809L
2 #include <sys/types.h>
3 #include <dirent.h>
4 #include <stdio.h>
5
6 int main();
7 void ls0();
```

ls0.h

```
1 #include "ls0.h"
2
3 int main() {
4     ls0();
5     return 0;
6 }
7
8 void ls0() {
9     DIR* directory = opendir(".");
10    struct dirent* entry;
11    while ((entry = readdir(directory)) != NULL) {
12        printf("%s\n", entry->d_name);
13    }
14    closedir(directory);
15 }
```

ls0.c

1.3. cp1

El último de los comandos obligatorios a realizar es *cp*. A diferencia del comando *cp0* propuesto en la Parte 2 de este mismo lab, esta vez debemos utilizar las syscalls mmap y memcpy.

Como mmap debe reservar memoria tanto para el archivo original como para la copia, se debe optimizar para estar a salvo de tener que copiar archivos muy grandes y no contar con la memoria necesarias.

Para esto se tuvo en cuenta la restricción que tiene el offset de mmap. Este offset debe ser un múltiplo del tamaño de página. Entonces lo propuesto es lo siguiente: en caso de que el tamaño de lo que falte copiar sea mayor al tamaño de página, se copian tantos bytes como ocupe la página, seteando el offset correspondiente. Caso contrario se copia lo faltante. De esta forma nos aseguramos de que el offset siempre sea un múltiplo entero del tamaño de página.

El código fuente de la solución es el siguiente:

```
1 #define _POSIX_C_SOURCE 200809L
2 #include <sys/mman.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7 #include <string.h>
8
9 int main(int argc, char* argv[]);
10 void cp1(char* origen, char* destino);
11 off_t get_file_size(char* fd);
```

cp1.h

```
1 #include "cp1.h"
2 #include <stdio.h>
3
4 int main(int argc, char* argv[]) {
5     if (argc != 3)
6         return -1;
7     cp1(argv[1], argv[2]);
8     return 0;
9 }
10
11 void cp1(char* origen, char* destino) {
12     char *input_mem, *output_mem;
13     off_t page_size = sysconf(_SC_PAGE_SIZE);
14     int input_fd = open(origen, O_RDONLY);
```

```
15      int output_fd = open(destino, O_RDWR|O_CREAT|O_TRUNC,
    ↪ S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH);
16      off_t tamano_total = get_file_size(origen);
17      ftruncate(output_fd, tamano_total);
18      off_t copiado = 0;
19      while (copiado < tamano_total) {
20          size_t a_copiar;
21          if (tamano_total - copiado >= page_size)
22              a_copiar = page_size;
23          else
24              a_copiar = tamano_total - copiado;
25          input_mem = mmap(NULL, a_copiar, PROT_READ, MAP_PRIVATE,
    ↪ input_fd, copiado);
26          output_mem = mmap(NULL, a_copiar, PROT_READ|PROT_WRITE,
    ↪ MAP_SHARED, output_fd, copiado);
27          memcpy(output_mem, input_mem, a_copiar);
28          munmap(input_mem, a_copiar);
29          munmap(output_mem, a_copiar);
30          copiado += a_copiar;
31      }
32      close(input_fd);
33      close(output_fd);
34  }
35
36  off_t get_file_size(char* file) {
37      struct stat buf;
38      stat(file, &buf);
39      return buf.st_size;
40  }
```

cp1.c

2. Comandos opcionales

2.1. ps0

El comando opcional a desarrollas en esta parte del lab es *ps*. En este caso se pide mostrar información de los procesos que se encuentran corriendo. Esta información es el identificado del proceso (pid) y el nombre del programa usado para lanzar dicho proceso.

En esta primera aproximación utiiizamos la información provista por el archivo comm dentro del directorio */proc/[pid]*. Dicho archivo contiene el nombre del programa buscado. De esta forma, ya contamos con toda la información necesaria para la resolución del problema.

El código fuente de la solución es el siguiente:

```h
#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <ctype.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define READ_BYTES 1024

int main();
void ps0();
int is_numeric(char* str);
void print_comm(char* pid);
void print_file(char* path);
```

ps0.h

```c
#include "ps0.h"

int main() {
    ps0();
    return 0;
}

void ps0() {
    DIR* directory = opendir("/proc");
    struct dirent* entry;
    printf("%5s %s\n", "PID", "COMMAND");
    while ((entry = readdir(directory)) != NULL) {
```

```
13          if (is_numeric(entry->d_name)) {
14              printf("%5s ", entry->d_name);
15              print_comm(entry->d_name);
16          }
17      }
18      closedir(directory);
19  }
20
21  int is_numeric(char* str) {
22      int i = 0;
23      while (str[i]) {
24          if (!isdigit(str[i]))
25              return 0;
26          i++;
27      }
28      return 1;
29  }
30
31  void print_comm(char* pid) {
32      char comm_path[6 + strlen(pid) + 5]; // "/proc/[pid]/comm"
33      memset(comm_path, '\0', 6 + strlen(pid) + 5);
34      sprintf(comm_path, "%s%s%s", "/proc/", pid, "/comm");
35      print_file(comm_path);
36  }
37
38  void print_file(char* path) {
39      int descriptor = open(path, O_RDONLY);
40      char buffer[READ_BYTES];
41      memset(buffer, '\0', READ_BYTES);
42      int leidos;
43      while ((leidos = read(descriptor, buffer, READ_BYTES)) > 0) {
44          printf("%s", buffer);
45          memset(buffer, '\0', READ_BYTES);
46      }
47      close(descriptor);
48  }
```

ps0.c

2.2. ps1

Una segunda versión del comando *ps* se propone en la sección *challenge del challenge*. En esta nueva versión, llamada ps1, se mostrará más información acerca de los procesos.

La información a mostrar debe ser obtenida del archivo stat dentro del directorio del proceso mediante *scanf*. La información elegida para mostrar es el pid, el programa que creó el proceso, el estado del proceso, el id del proceso padre (ppid), el grupo del proceso y la sesión sobre el cual fue creado.

Esta versión del comando es equivalente a ejecutar *ps -eo pid,comm,state,ppid,pgrp,session*.

El código fuente de la solución es el siguiente:

```c
#define _POSIX_C_SOURCE 200809L

int main();
void ps0();
int is_numeric(char* str);
void print_stat(char* pid);
void print_line(char* file);
```

ps1.h

```c
#include "ps1.h"
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <ctype.h>
#include <string.h>
#include <unistd.h>

int main() {
    ps0();
    return 0;
}

void ps0() {
    DIR* directory = opendir("/proc");
    struct dirent* entry;
    printf("%5s %-16s %5s %5s %5s %6s\n", "PID", "COMMAND", "STATE",
    ↪ "PPID", "GROUP", "SESSION");
    while ((entry = readdir(directory)) != NULL) {
        if (is_numeric(entry->d_name)) {
            print_stat(entry->d_name);
        }
    }
    closedir(directory);
}

int is_numeric(char* str) {
    int i = 0;
    while (str[i]) {
        if (!isdigit(str[i]))
            return 0;
        i++;
    }
    return 1;
}

```

```c
void print_stat(char* process) {
    char comm_path[6 + strlen(process) + 5]; // "/proc/[pid]/comm"
    memset(comm_path, '\0', 6 + strlen(process) + 5);
    sprintf(comm_path, "%s%s%s", "/proc/", process, "/stat");
    print_line(comm_path);
}

void print_line(char* comm_path) {
    int pid, ppid, pgrp, session;
    char comm[16], state;
    FILE* file = fopen(comm_path, "r");
    fscanf(file, "%d %*[(]%[^)]%*[)] %c %d %d %d", &pid, comm,
    ↪ &state, &ppid, &pgrp, &session);
    fclose(file);
    printf("%5d %-16s %5c %5d %5d %6d\n", pid, comm, state, ppid,
    ↪ pgrp, session);
}
```

ps1.c