Unix:
Entrada – Salida
Archivos
Directorios
Información de Archivos

Sistema Unix

- El sistema Unix proporciona sus servicios a través de un conjunto de llamadas al sistema:
  - funciones que están dentro del sistema operativo y pueden ser invocadas por programas
- Todas las entradas y salidas se realizan por medio de la lectura o escritura de archivos
  - Los dispositivos periféricos son considerados archivos en el sistema: /dev/audio
- Antes de leer o escribir se debe informar al sistema, el cual verifica los permisos

Sistema Unix

- Al abrir un archivo se recibe un “file descriptor” (fd), que es un entero
  - fd identifica a un archivo, tal como un “file pointer”
- Todo programa en Unix parte con 3 “fd” ya abiertos:
  - 0 entrada estándar
  - 1 salida estándar
  - 2 salida de errores
- Se puede redirigir la entrada y salida con “<” y “>”

Entrada - Salida

- Operaciones básicas de entrada y salida:

```
nread = read(fd, buf, nbytes);
nwritten = write(fd, buf, nbytes);
```

- Cada llamada retorna el número de bytes transferidos
- En el caso de “write” es un error que éste no sea igual al número pedido
- En caso de fin de archivo se retorna un cero y, en caso de error, un -1

Entrada - Salida

- Copiar la entrada en la salida:

```
#define BUFSIZE 4096
main(){
  char buf[BUFSIZE];
  int n;

  while((n=read(0, buf, BUFSIZE))>0)
    write(1, buf, n);
}
```

- Ejemplo: copia.c

José Urzúa R jourzua@dcc.uchile.cl

Entrada - Salida

- Podríamos construir nuestro propio getchar:

```
int getchar(){
  char c;

  return (read(0, &c, 1)==1? (unsigned char)c :
  EOF);
}
main(){
  int c;
  while((c=getchar())!=EOF)
    putchar(c);
}
```

getchar1.c

José Urzúa R jourzua@dcc.uchile.cl

Entrada - Salida

- Una versión más eficiente utilizando un buffer:

```
int getchar(){
  static char buf[BUFSIZ];
  static char *p=buf;
  static int n=0;
```

getchar2.c

```
  if(n==0){ /* buffer vacío */
    n=read(0,buf, sizeof buf);
    p=buf;
  }

  return (--n>=0? (unsigned char)*p++ : EOF);
}
```


José Urzúa R jourzua@dcc.uchile.cl

Entrada - Salida

- Al utilizar un buffer de tipo “static”, nos aseguramos que exista aunque se cierre la función
- Si “getchar” está definido en dentro de alguna biblioteca que se está utilizando, se debe agregar al inicio:

```
#undef getchar
```


José Urzúa R jourzua@dcc.uchile.cl

Llamada a Sistema vs librerías de I/O

Llamadas a Sistema	**Estándar I/O**
open	fopen
close	fclose
read/write	getchar/putchar
	getc/putc
	fgetc/fputc
	fread/fwrite
	gets/puts
	fgets/fputs
	scanf/printf
	fscanf/fprintf
lseek	fseek

Archivos

- Para abrir archivos existen dos funciones:

```
int open(const char *path, int flags, mode_t modo);
int open(const char *path, int flags);
int creat(const char *path, mode_t modo);
```

- “open” funciona similar a “fopen”, los “flags” más comunes son:
  - O_RDONLY: sólo lectura
  - O_WRONLY: sólo escritura
  - O_RDWR: lectura y escritura

Archivos

- Los “flags” anteriores están definidos en <fcntl.h> en sistemas Unix System V y en <sys/file.h> en versiones Berkeley (BSD)
- Abrir archivo ya existente para lectura:

```
fd = open(path, O_RDONLY);
```

- Es un error abrir un archivo que no existe para lectura
- Para crear archivos o reescribir alguno existente:

```
fd = creat(path, perm);
```


Archivos

- En Unix hay 9 bits para mantener la información de permisos de un archivo:
  - Controlan lectura, ejecución y escritura, para el dueño, el grupo del dueño y el resto de los usuarios
  - Es conveniente manejar los permisos con un número octal de tres digitos:

```
owner   group   other
r w x   r w x   r w x
4 2 1   4 2 1   4 2 1
```

  - 755: todos los permisos para el dueño y leer y ejecutar para los demás

Archivos

- Ejemplo de copiar un archivo: my_cp.c
- Para borrar un archivo se utiliza:

```
int unlink(char *path);
```

- Si los permisos lo permiten y ningún proceso tiene el archivo abierto, es borrado
- Si algún proceso tiene el archivo abierto, no será borrado hasta que el último “fd” sea cerrado
- Devuelve 0 en caso de éxito y -1 en caso de error

Archivos

- Posicionarse en un archivo:

```
long lseek(int fd, long offset, int origen);
```

- Similar a “fseek”, excepto por el primer argumento (FILE *) y el valor de retorno (!= 0 en caso de error)
- Retorna la nueva posición dentro del archivo, o -1 si hubo un error
- Origen:
  - 0: desde el comienzo
  - 1: a partir de la posición actual
  - 2: desde el fin de archivo

Archivos

- “offset” puede ser positivo o negativo

```
lseek(fd, 0L, SEEK_EOF);  /* para append */
lseek(fd, 0L, SEEK_SET);  /* rewind */
```

- Los “flag” para abrir un archivo se pueden combinar con las siguientes macros:
  - O_CREAT: permite crear un archivo si no existe
  - O_TRUNC: trunca el archivo a largo cero
  - O_APPEND: indica que se vaya al final del archivo, antes de cada escritura

Archivos

- La llamada:

```
creat(path, mode);
```

equivale a:

```
open(path, O_WRONLY | O_CREAT | O_TRUNC, mode);
```

Directorios

- En Unix, un directorio es un archivo que contiene una lista de archivos en su interior
- Dicha lista contiene indicaciones de donde se localizan los archivos: el índice en la tabla de i-nodos
- El i-nodo mantiene toda la información de un archivo, excepto el nombre
- Un elemento de la lista que tiene un directorio, tiene 2 ítemes: el nombre de archivo y el i-nodo
- El formato y contenido de un directorio puede variar en distintos sistemas

José Urzúa R jourzua@dcc.uchile.cl

Directorios

- Para el manejo de directorios se utiliza el header <dirent.h>:
  - Define una estructura que es usada para obtener los nombres de archivo de un directorio (struct_dirent)
  - Esta estructura contiene el nombre de archivo y el i-nodo correspondiente
- Abrir un directorio:

```
DIR *dirp;

dirp = opendir(dname); /* NULL para error */
```

- “dname” debe ser un directorio

José Urzúa R jourzua@dcc.uchile.cl

Directorios

- Leer un directorio:

```
struct dirent *d;
d=readdir(dirp); /* entrega siguiente nombre de
                    archivo en d->d_name,
                    retorna NULL al final */
```

- “dirp” es el puntero retornado por “opendir()”
- Cuando se encuentra el final del directorio, retorna NULL, asignando el error en “errno”
- En caso de error, retorna NULL, dejando en “errno” el valor correspondiente al error

José Urzúa R jourzua@dcc.uchile.cl

Directorios

- Cerrar un directorio:

```
int closedir(DIR *dirp);
```

- Es usado para indicar que ya concluimos la lectura del directorio
- Retorna 0 en caso de éxito, o -1 en caso de error
- Volver al inicio el puntero:

```
void rewinddir(DIR *dirp);
```


José Urzúa R jourzua@dcc.uchile.cl

Directorios

- Obtener el directorio actual:

```
buf = getcwd(char *buf, long size);
```

- Almacena el nombre del directorio actual en buf, retorna el valor de buf o NULL en caso de error
- “size” indica el tamaño máximo soportado para “buf”, si “buf” es más grande, retorna NULL
- cambiar de directorio:

```
int chdir(char *path);
```


Directorios

- En caso de éxito “chdir” retorna 0, en error retorna -1
- Crear directorios:

```
int mkdir(char *pathname, mode_t mode);
```

- Crea un directorio de nombre “pathname”, con los permisos especificados en “mode”:
  - retorna 0 en caso de éxito, -1 en caso de error
- Borrar directorios:

```
int rmdir(char *pathname);
```


Directorios

- Para poder borrar un directorio, este debe estar vacío
- En caso de éxito retorna 0, en caso de error -1
- No funciona para directorios:

```
int link(char *oldpath, char *newpath);
```

- Crea un “enlace físico” al nombre de archivo “oldpath”
- Si el nuevo nombre existe, no será sobreescrito
- Ambos nombres se refieren al mismo archivo, con los mismo permisos, es imposible saber que nombre era el original

Directorios

- Cambiar nombre:

```
int rename(char *oldpath, char *newpath);
```

- Crea un nuevo enlace a un archivo existente y borra el enlace antiguo
- Si “oldpath” y “newpath” se refieren al mismo archivo, no realiza ningún cambio
- Se puede cambiar nombres de directorios, pero no siempre archivos bajo directorios distintos

Información Archivos

- Para averiguar atributos de un archivo se utiliza:

```
int stat(char *fname, struct stat *buf);
```

- “stat” obtiene la información del archivo “fname” y la asigna a la estructura “buf”
- La estructura “stat” es definida en: “<sys/stat.h>”
- Retorna 0 en caso de éxito y -1 en caso de error, dejando el valor correspondiente en errno

Información Archivos

- La estructura “stat” tiene campos como:

st_mode	modo (=perms)
st_ino	inode
st_dev	device id
st_nlink	número de links
st_uid	user id del owner
st_gid	grupo
st_size	tamaño del archivo en bytes
st_atime	dia y hora del último acceso
st_ctime	día y hora del último cambio (p.ej. perms)
st_mtime	día y hora de la última modificación

Información Archivos

- Hay macros para "testear"el modo:

```
S_ISDIR             directorio?
S_ISCHR             char special file?
S_ISBLK             block special file?
S_ISREG             regular file?
S_ISFIFO            named pipe?
```

- Ejemplo:

```
if(!S_ISDIR(status.st_mode))
```


Información Archivos

- Los atributos de un archivo se pueden modificar con:

```
int chmod(char *path, mode_t mode);
int chown(char *path, uid_t owner, gid_t group);
```

- “chmod” cambia los permisos del archivo “path” a “mode”
  - Retorna 0 en caso de éxito, -1 en caso de error
- “chown” cambia el dueño del archivo “path”, al dueño y grupo especificado:
  - Retorna 0 en caso de éxito, -1 en caso de error