Preparación para la entrevista de estructuras de datos, algoritmos , práctica por tema, C++, Java , Python

Llamadas al sistema de entrada-salida en C | Crear, Abrir, cerrar, leer, escribir

Nivel de dificultad: Medio • Última actualización: 28 de octubre de 2021

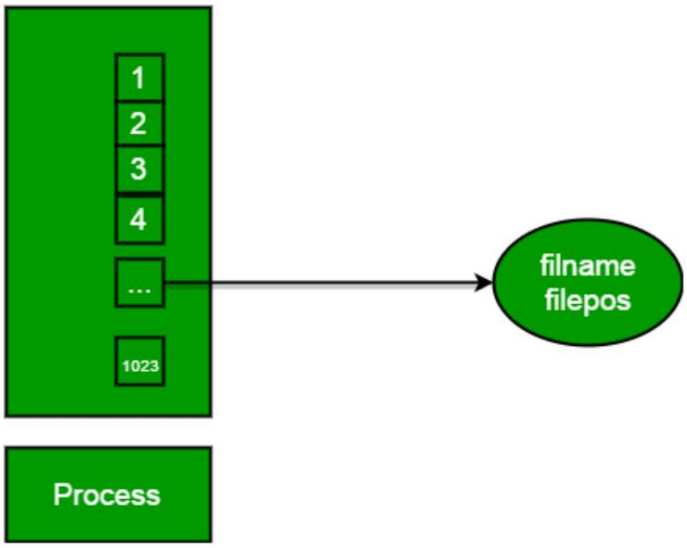
Terminología importante

¿Qué es el descriptor de archivo?

El descriptor de archivo es un número entero que identifica de forma única un archivo abierto del proceso.

Tabla de descriptores de archivos: La tabla de descriptores de archivos es un conjunto de índices de matrices de enteros que son descriptores de archivos, cuyos elementos apuntan a entradas de la tabla. El sistema operativo proporciona una tabla de descriptores de archivos única para cada proceso.

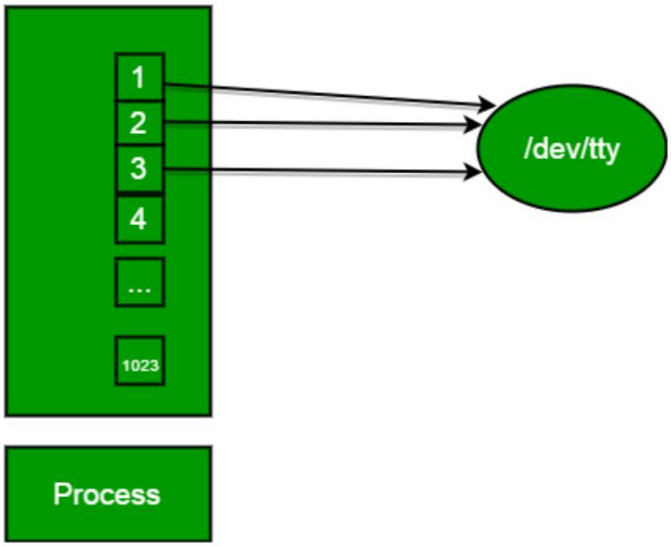
Entrada de la tabla de archivos: las entradas de la tabla de archivos son una estructura en memoria sustituta de un archivo abierto, que se crea cuando el proceso solicita abrir un archivo y estas entradas mantienen la posición del archivo.



Descriptores de archivo estándar: cuando se inicia cualquier proceso, los descriptores de archivo fd 0, 1 y 2 de la tabla de descriptores de archivo de ese proceso se abren automáticamente. (De manera predeterminada) cada uno de estos 3 fd hace referencia a la entrada de la tabla de archivos de un archivo llamado /dev/tty

/dev/tty: sustituto en memoria para la terminal

Terminal: Combinación de teclado y pantalla de vídeo



Leer desde stdin => leer desde fd 0: Siempre que escribimos cualquier carácter

Desde el teclado, lee desde stdin hasta fd 0 y guarda en un archivo llamado /dev/tty.

Escribir en stdout => escribir en fd 1: Siempre que vemos alguna salida en la pantalla de video, es desde el archivo llamado /dev/tty y se escribe en stdout en la pantalla a través de fd 1.

Escribir en stderr => escribir en fd 2: Vemos cualquier error en la pantalla de video, también es desde ese archivo escrito en stderr en la pantalla a través de fd 2.

Llamadas al sistema de E/S

Básicamente, hay un total de 5 tipos de llamadas al sistema de E/S:

1. Crear: se utiliza para crear un nuevo archivo vacío.

Sintaxis en lenguaje C: int
create(char *filename, mode_t mode)

Parámetro:

- nombre de archivo: nombre del archivo que desea crear • modo: indica los permisos del nuevo archivo.

Devoluciones:

- Devuelve el primer descriptor de archivo no utilizado (generalmente 3 cuando se crea por primera vez en el proceso porque 0, 1, 2 fd están reservados)
- devolver -1 cuando hay error

Cómo funciona en el sistema operativo

- Crear un nuevo archivo vacío en el disco • Crear una entrada en la tabla de archivos • Establecer el primer descriptor de archivo no utilizado para que apunte a la entrada de la tabla de archivos • Devolver el descriptor de archivo usado, -1 en caso de error

2. abrir: se utiliza para abrir el archivo para lectura, escritura o ambas.

Sintaxis en lenguaje C

```
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>
int abierto (const char* Ruta, int banderas [, int modo ]);
```

Parámetros

- Ruta: ruta al archivo que desea utilizar
  - utilice una ruta absoluta que comience con "/", cuando no esté trabajando en el mismo directorio del archivo.
  - Utilice la ruta relativa, que es solo el nombre del archivo con extensión, cuando funcionan en el mismo directorio de archivos.
- banderas: como te gusta usarlas
  - O_RDONLY: solo lectura, O_WRONLY: solo escritura, O_RDWR: lectura y escritura, O_CREAT: crear archivo si no existe, O_EXCL: evitar la creación si ya existe

Cómo funciona en el sistema operativo

• Buscar el archivo existente en el disco • Crear una entrada en la tabla de archivos • Establecer el primer descriptor de archivo no utilizado para que apunte a la entrada de la tabla de archivos • Devolver el descriptor de archivo usado, -1 en caso de error

---

do

```
// Programa en C para ilustrar // llamada al
sistema abierto #include<stdio.h>

#include<fcntl.h>
#include<errno.h>
externo int errno; int main() {
```

```
// si el archivo no está en el directorio // entonces se crea el
archivo foo.txt. int fd = open("foo.txt", O_RDONLY |
O_CREAT);

printf("fd = %d/n", fd);

if (fd ==-1) { // imprime
qué
tipo de error tiene un código printf(" Número de error % d\n", errno);


// imprimir detalle del programa "Éxito o fracaso" perror("Programa"); } return 0; }
```

Producción:

```
fd = 3
```

3. cerrar: le dice al sistema operativo que ha terminado con un descriptor de archivo y cierra el archivo señalado por fd.

Sintaxis en lenguaje C

```
#include <fcntl.h>
int cerrar(int fd);
```

Parámetro:

- fd : descriptor de archivo

Devolver:

- 0 en caso de éxito.
- -1 en caso de error.

Cómo funciona en el sistema operativo

- Destruir la entrada de la tabla de archivos referenciada por el elemento fd del descriptor de archivo

mesa

– ¡Siempre que ningún otro proceso lo esté señalando! • Establezca

el elemento fd de la tabla de descriptores de archivos en NULL

---

do

```
// Programa en C para ilustrar el cierre del sistema Llamada
#include<stdio.h> #include
<fcntl.h> int main() { int fd1 =
open("foo.txt",

O_RDONLY); if (fd1 < 0) { perror("c1"); exit(1); } printf("se abrió el
fd = % d\n", fd1);



// Usando la llamada al sistema de cierre if
(close(fd1) < 0) { perror("c1"); exit(1); }

printf("cerró el fd.\n"); }
```

Producción:

```
abrió el fd = 3
cerró el fd.
```

---

do

```
// Programa en C para ilustrar el cierre del sistema. Llamada
#include<stdio.h>
#include<fcntl.h>
```

```
int main() { //

supongamos que foo.txt ya está creado int fd1 = open("foo.txt",
O_RDONLY, 0); close(fd1);


// supongamos que baz.tzt ya está creado int fd2 = open("baz.txt",
O_RDONLY, 0);

printf("fd2 = % d\n", fd2); salir(0); }
```

Producción:

```
fd2 = 3
```

Aquí, en este código, la primera función open() devuelve 3 porque, al crear el proceso principal, los descriptores de archivo 0, 1 y 2 ya estaban ocupados por stdin, stdout y stderr. Por lo tanto, el primer descriptor de archivo no utilizado es 3 en la tabla de descriptores de archivo. Después, la llamada al sistema close() libera estos descriptores de archivo y los establece como nulos. Por lo tanto, al ejecutar la segunda función open(), el primer descriptor de archivo no utilizado también es 3. Por lo tanto, la salida de este programa es 3.

4. lectura: desde el archivo indicado por el descriptor de archivo fd, la función read() lee cnt bytes de entrada en el área de memoria indicada por buf.
Una lectura exitosa actualiza el tiempo de acceso al archivo.

Sintaxis en lenguaje C

```
size_t read(int fd, void* buf, size_t cnt);
```

Parámetros:

- fd: descriptor de archivo
- buf: búfer desde el que leer datos
- cnt: longitud del buffer

Devuelve: cuántos bytes se leyeron realmente

• devolver el número de bytes leídos en caso de éxito • devolver 0 al llegar al final del archivo
• devolver -1 en caso de error
• devolver -1 en caso de interrupción de la señal

Puntos importantes

• buf debe apuntar a una ubicación de memoria válida con una longitud no menor que el tamaño especificado debido al desbordamiento. • fd debe ser un descriptor de archivo válido devuelto desde open() para realizar operación de lectura porque si fd es NULL entonces la lectura debería generar un error.
• cnt es el número solicitado de bytes leídos, mientras que el valor de retorno es el número real de bytes leídos. Además, a veces, la llamada al sistema de lectura debería leer menos bytes que cnt.

---

do

```
// Programa en C para ilustrar // lectura del
sistema Llamada
#include<stdio.h>
#include <fcntl.h>
int main() { int

fd, sz; char *c
= (char *) calloc(100, sizeof(char));

fd = abrir("foo.txt", O_RDONLY); si (fd < 0)
{ perror("r1"); salir(1); }

sz = read(fd, c, 10);
printf("llamado read(% d, c, 10). devolvió que" " %d bytes fueron
leídos.\n", fd, sz); c[sz] = '\0'; printf("Esos bytes son los
siguientes: % s\n",
c); }
```

Producción:

llamado read(3, c, 10). devolvió que se leyeron 10 bytes.
Estos bytes son los siguientes: 0 0 0 foo.

Supongamos que foobar.txt consta de los 6 caracteres ASCII “foobar”.

¿Entonces cuál es el resultado del siguiente programa?

---

do

```
// Programa en C para ilustrar
// leer llamada al sistema
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>

int principal()
{
carácter c;
int fd1 = open("muestra.txt", O_RDONLY, 0);
int fd2 = open("muestra.txt", O_RDONLY, 0);
leer(fd1, &c, 1);
leer(fd2, &c, 1);
printf("c = %c\n", c);
salir(0);
}
```

Producción:

c = f

Los descriptores fd1 y fd2 cada uno tiene su propia entrada de tabla de archivos abiertos, por lo que cada descriptor tiene su propia posición de archivo para foobar.txt . Por lo tanto, la lectura de fd2 lee el primer byte de foobar.txt , y la salida es c = f, no c = o.

5. escribir: escribe bytes cnt desde buf al archivo o socket asociado con

fd.cnt no debe ser mayor que INT_MAX (definido en el archivo de encabezado limits.h).

Si cnt es cero, write() simplemente devuelve 0 sin intentar ninguna otra acción.

```
#include <fcntl.h>
tamaño_t escribir (int fd, void* buf, tamaño_t cnt);
```

Parámetros:

- fd: descriptor de archivo
- buf: búfer para escribir datos
- cnt: longitud del buffer

Devuelve: cuántos bytes se escribieron realmente

- devuelve el número de bytes escritos en caso de éxito •

devuelve 0 al llegar al final del archivo

- devolver -1 en caso de error
- devolver -1 en caso de interrupción de la señal

Puntos importantes

- El archivo debe estar abierto para operaciones de escritura •

buf debe tener al menos la longitud especificada por cnt porque si el tamaño de buf
Si es menor que cnt, entonces buf provocará una condición de desbordamiento.

- cnt es el número de bytes solicitados para escribir, mientras que el valor de retorno es el número real de bytes escritos. Esto ocurre cuando fd tiene menos bytes para escribir que cnt.
- Si write() es interrumpido por una señal, el efecto es uno de los siguientes:

-Si write() aún no ha escrito ningún dato, devuelve -1 y establece errno en EINTR.

-Si write() ha escrito exitosamente algunos datos, devuelve el número de bytes que escribió antes de ser interrumpido.

do

```
// Programa en C para ilustrar
// escribir llamada al sistema
#include<stdio.h>
#include <fcntl.h>
principal()
{
entero tamaño;

int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
si (fd < 0)
{
perror("r1");
salir(1);
}

sz = write(fd, "hola geeks\n", strlen("hola geeks\n"));

printf("llamado write(% d, \"hola geeks\\n\", %d)."
" Devolvió %d\n", fd, strlen("hola geeks\n"), sz);

cerrar(fd);
}
```

Producción:

Se llamó a write(3, "hola geeks\n", 12). Devolvió 11.

Aquí, cuando ves en el archivo foo.txt después de ejecutar el código, obtienes un " Hola geeks ". Si el archivo foo.txt ya tiene algún contenido, escriba La llamada al sistema sobrescribe el contenido y todo el contenido anterior es y eliminado solo " Hola geeks " contenido que tendrá el archivo.

Imprima “hola mundo” desde el programa sin usar printf ni cout función.

do

```
// Programa en C para ilustrar // Llamadas
al sistema de E/S
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<fcntl.h>

int main (void) { int fd[2];

char buf1[12] = "hola
mundo"; char buf2[12];


// supongamos que foobar.txt ya está creado fd[0] = open("foobar.txt",
O_RDWR); fd[1] = open("foobar.txt", O_RDWR);


escribir(fd[0], buf1, strlen(buf1)); escribir(1, buf2, leer(fd[1], buf2, 12));


cerrar(fd[0]); cerrar(fd[1]);


devuelve 0; }
```

Producción:

Hola Mundo

En este código, la cadena fd[0] "Hola Mundo" Es la primera escritura en la entrada estándar del array buf1 se escribe en la entrada estándar del array buf2. Posteriormente, se escribe en la salida estándar del array buf2 e imprime " Hola Mundo ". la salida. Este artículo es una contribución de Kadam Patel. Si te gusta GeeksforGeeks y te gustaría contribuir, también puedes escribir un artículo usando write.geeksforgeeks.org O envía tu artículo a review-team@geeksforgeeks.org. Publica tu artículo en la página principal de GeeksforGeeks y ayuda a otros geeks.

Por favor, escriba comentarios si encuentra algo incorrecto o desea compartir más información sobre el tema tratado anteriormente.

Anterior

Llamada al sistema Linux dup() y dup2()

Próximo

Mutex vs Semáforo

ARTÍCULOS RECOMENDADOS

Página: 1 2 3

01 lseek() en C/C++ para leer el byte n alternativo y escribirlo en otro archivo

01, abril 17

02 Leer/escribir estructura en un archivo en do

28 de junio de 2017

03 ¿El compilador de C++ crea un constructor predeterminado cuando lo usamos? ¿Escribir el nuestro?

28 de julio de 2010

04 Implementa tu propia cola (Lee las últimas n líneas de un archivo enorme)

06, 16 de mayo

05 Cómo leer e imprimir un Valor entero en C

05, dic. 18

06 Cómo leer un formato PGMB imagen en C

18 de enero de 2021

07 Programa en C para leer un rango de bytes de un archivo e imprimirlo en la consola

25 de junio de 2021

08 ¿Cómo ingresar o leer un carácter, una palabra y una oración del usuario en C?

21 de septiembre de 2021

Artículo aportado por:

Geeks para Geeks

Votar por la dificultad

Dificultad actual: Media

Fácil | Normal | Medio | Duro | Experto

Mejorado por: ritwikshanker, srinam, gabaa406, anikaseth98, ruhelaa48, vivekjoshi556, avtarkumar719

Etiquetas del artículo: programación de sistemas, Lenguaje C

Mejorar el artículo | Informar de un problema

¿Escribes código en un comentario? Usa ide.geeksforgeeks.org, generar enlace y compartir el enlace aquí.

Cargar comentarios

GeeksforGeeks

A-143, Piso 9, Torre Corporativa Sovereign,
Sector 136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

JavaScript

Oreja

ReactJS

NodeJS

Escribe la experiencia de la entrevista

Pasantías

Pasantía de video