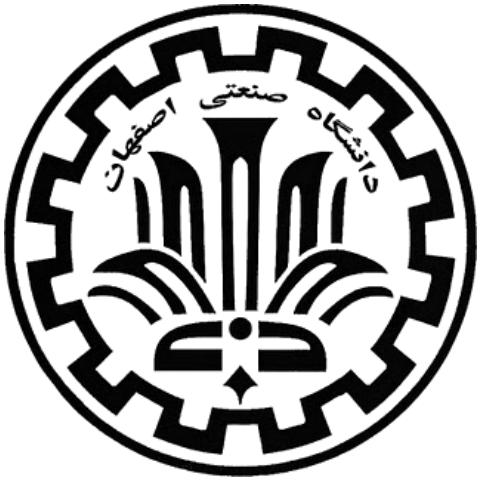

دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

پیش‌گزارش جلسه سوم

پاییز ۱۴۰۲

فهرست مطالب

1 فراخوانی سیستمی **SYSTEM CALL** 2

2 **LIBRARY** ها 7

3 ساخت و استفاده از **STATIC LIBRARY** و **DYNAMIC LIBRARY** 8

4 مثال‌ها 10

1 فراخوانی سیستمی System Call

دستورات دریافت شده توسط پوسته به هسته ارسال می شوند. این دستورات در قالب فراخوانی های سیستمی به هسته ارسال شده و در آنجا به فراخوانی های هسته تبدیل می شوند. در نهایت این فراخوانی های هسته اند که عملیات مشخص شده را به انجام می رسانند.
man syscalls را اجرا کنید تا اطلاعات خوبی راجع به systemcall ها کسب کنید.
هشت دسته از مهمترین فراخوانی های سیستم در استاندارد POSIX عبارتند از:

1. مدیریت فایل ها
2. مدیریت فایل سیستم و دایرکتوری ها
3. مدیریت سطح دسترسی و حفاظت از فایل ها
4. مدیریت زمان
5. مدیریت فرآیندها
6. مدیریت سیگنال ها (signal)
7. مدیریت سوکت ها (socket)
8. مدیریت حافظه مشترک (shared memory)

در این آزمایشگاه با بخش عمده این فراخوانی‌های سیستمی کار خواهید کرد. در این جلسه فراخوانی های سیستمی برای مدیریت فایل ها، مدیریت فایل سیستم و مدیریت سطح دسترسی بررسی می شوند. برای هر یک از فراخوانی‌های سیستمی که در این فایل معرفی شده است، با اجرای man می‌توانید اطلاعات دقیقتر و کاملتری کسب کنید.

1.1 فراخوانی های سیستمی برای مدیریت فایل ها

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Open and possibly create a file or device

flags:

O_RDONLY	open for reading only
O_WRONLY	open for writing only
O_RDWR	open for reading and writing
O_NONBLOCK	do not block on open
O_APPEND	append on each write
O_CREAT	create file if it does not exist
O_TRUNC	truncate size to 0
O_EXCL	error if create and file exists

modes:

S_ISUID	04000	set user ID on execution
S_ISGID	02000	set group ID on execution
S_ISVTX	01000	`sticky bit'
S_IRWXU	00700	read, write, execute by owner
S_IRUSR	00400	read by owner
S_IWUSR	00200	write by owner
S_IXUSR	00100	execute (search on directory) by owner
S_IRWXG	00070	read, write, execute by group
S_IRGRP	00040	read by group
S_IWGRP	00020	write by group
S_IXGRP	00010	execute (search on directory) by group
S_IRWXO	00007	read, write, execute by others
S_IROTH	00004	read by others
S_IWOTH	00002	write by others
S_IXOTH	00001	execute (search on directory) by others

IS_UID: وقتی فایلی که این بیت برایش یک شده است اجرا میشود فایل با دسترسی یوزری که آن را اجرا کرده اجرا نمیشود بلکه با دسترسی مالکش اجرا می شود. فقط روی فایلها اثر میکند و روی دایرکتوری اثر نمیکند.

```
int close(int fd);
```
Close a file descriptor

```
ssize_t read(int fd, void *buf, size_t count);
```
Read data from a file into a buffer

```
ssize_t write(int fd, const void *buf, size_t count);
```
Write data from a buffer into a file

```
off_t lseek(int fd, off_t offset, int whence);
```
Move the file pointer

```
int stat(const char *path, struct stat *buf);
```
Get a file's status information

```
int fstat(int fd, struct stat *buf);
```
Get a file's status information

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```
dup system call allocates a new file descriptor that refers to the same open file description as the descriptor oldfd. The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd.

```
int access(const char *path, int amode)
```
Check a file's accessibility

```
int rename(const char *old, const char *new)
```
Give a file a new name

```
int fcntl(int fildes, int cmd, ...)
```
File locking and other operations

1.2 فراخوانی های سیستمی برای مدیریت فایل سیستم و دایرکتوری ها

```
int mkdir(const char *pathname, mode_t mode);
```
Creates a directory in path

```
int rmdir(const char *pathname);
```
Remove an empty directory

```
int link(const char *name1, const char *name2)
```
link() creates a new link named name2 to an existing file with name1

```
int unlink(const char *name);
```
unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is also deleted.

```
int chdir(const char *path);
int fchdir(int fd);
```
Change working directory

1.3 فراخوانی های سیستمی برای مدیریت سطح دسترسی و حفاظت از فایل ها

```
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
```
Change a file's protection bits, Only the owner of a file (or the super-user) may change the mode.

```
uid_t getuid(void);
```
getuid() returns the real user ID of the calling process.

```
uid_t geteuid(void);
```
geteuid() returns the effective user ID of the calling process. Effective UserID is normally the same as real UserID, but sometimes it is changed to enable a non-privileged user to access files that can only be accessed by a privileged user like root.

```
int setuid(uid_t uid);
```
setuid() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set.

```
int seteuid(uid_t euid);
```
Sets effective uid of the calling process

```
int setegid(gid_t egid);
```
Set the caller's uid

```
gid_t getgid(void)
```
Get the caller's gid

```
gid_t getegid(void)
```
Set effective group ID

```
set effective group ID
int setegid(gid_t gid);
int setgid(gid_t gid);
```
Set the caller's gid

```
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
```

chown() changes the ownership of the file specified by path, which is dereferenced if it is a symbolic link.

```
mode_t umask(mode_t mask);
```

umask() sets the calling process's file mode creation mask (umask) to mask & 0777 (only the file permission bits of mask are used), and returns the previous value of the mask.

2 Library ها

همانطور که میدانید بسیاری اوقات برنامه‌نویسان مستقیم از فراخوانی‌های سیستمی استفاده نمی‌کنند، بلکه از کتابخانه‌های زبان‌های برنامه‌نویسی استفاده می‌کنند. در پیاده‌سازی این کتابخانه‌ها هرجا نیاز باشد از فراخوانی‌های سیستمی استفاده شده‌است.

2.1 C Library <stdio.h>

در اینجا تعدادی کتابخانه‌های پرکاربرد C که می‌تواند برای کار با فایل‌ها (به جای فراخوانی مستقیم system call ها) استفاده شود معرفی می‌شوند.

printf

```
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
```

The function printf() writes output to stdout (the standard output stream), fprintf() writes output to the given output stream, sprintf(), snprintf() write to the character string str. The function snprintf() writes at most *size* bytes (including the terminating null byte ('\0')) to *str*.

scanf

```
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
```

Input format conversion

2.2 C Library <string.h>

strcmp

```
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

The strcmp() function compares the two strings s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

The strncmp() function is similar, except it only compares the first (at most) n bytes of s1 and s2.

strlen

```
size_t strlen(const char *s);
```

The strlen() function calculates the length of the string *s*, excluding the terminating null byte ('\0').

3 ساخت و استفاده از static library و dynamic library

همه کتابخانه‌های استاندارد در زبان c دارای پیشوند lib هستند که هنگام استفاده معمولاً به عنوان flag در gcc ذکر نمی‌شوند. اما در موارد دیگر، نیاز است کتابخانه در دستور کامپایل معرفی شود. نحوه ایجاد کتابخانه‌ها یا کامپایل با کتابخانه‌های مشخص در ادامه شرح داده می‌شود.

3.1 Static library

پسوند این دسته از فایل‌ها a. است و تنها در هنگام کامپایل نیاز می‌شوند. هر کتابخانه در واقع از مجموعه‌ای از توابع یا کلاسها تشکیل شده‌است. برای ایجاد یک static library یک فایل c ایجاد کرده و توابع موردنظر خود را در آن پیاده‌سازی می‌کنیم. این توابع می‌توانند در یک فایل c یا تعداد بیشتری فایل پیاده‌سازی شوند. سپس این فایل‌ها را کامپایل کرده و فایل‌های object آن را ایجاد می‌کنیم.

```
gcc -c staticlib.c -o staticlib.o
```

حال کافی است که مجموعه این فایل‌ها را در یک فایل با پسوند a. ذخیره کنیم. برای این کار از دستور ar استفاده می‌کنیم که معمولاً برای فشرده‌سازی، تصحیح و استخراج فایل‌ها از آن استفاده می‌شود:

```
ar -r libstaticLib.a libstaticLib.o
```

حال فرض کنید کدی به اسم myapp.c داریم که از توابع موجود در این static library استفاده کرده است. برای اینکه کد myapp.c از توابع کتابخانه ما استفاده کند، لازم است یک فایل header با پسوند h. بسازیم که شامل الگو یا امضای (نام تابع همراه با آرگومان‌های ورودی و خروجی بدون پیاده‌سازی تابع) توابع پیاده‌سازی شده در کتابخانه ما است. برای اینکه این کتابخانه را به صورت استاتیک به کد خود لینک کنیم به صورت زیر کدمان را کامپایل می‌کنیم:

```
gcc myapp.c -L./ -lstaticLib -o app.out
```

دقت کنید که flagهای l- و L- باید بعد از اسم برنامه نوشته شوند. زیرا ابتدا باید برنامه شما آماده باشد تا لینکر بتواند کتابخانه‌های مورد نیاز را تشخیص دهد و به آن لینک کند. در ادامه توضیح کوتاهی در مورد هرکدام بیان می‌شود.

-llibrary : قراردادن این عبارت باعث می‌شود که لینکر تمام مسیرهای استاندارد (برای مثال /usr/lib/) را برای پیدا کردن کتابخانه‌ای به اسم liblibrary.a جستجو کند و سپس برنامه را با آن لینک کند. برای مثال در دستور زیر:

```
gcc myapp.c -lpthread
```

لینکر به دنبال فایلی با نام libpthread.a می‌گردد.

-Lpath : مسیر داده شده را به لیست مسیرهایی که l- در آن جستجو می‌کند اضافه می‌کند.

3.2 Dynamic library

وقتی از یک کتابخانه static در برنامه خود استفاده می‌کنیم، فایل‌های باینری (آبجکت) کتابخانه کپی شده و به فایل‌های آبجکت برنامه ما link می‌شود. بنابراین برنامه قابل اجرای نهایی ما شامل فایل‌های کتابخانه می‌باشد. اما وقتی از کتابخانه های dynamic استفاده می‌کنیم کد آنها به صورت جداگانه برای برنامه ما کپی نمی‌شود بلکه به صورت shared برای همه برنامه‌هایی که توابع کتابخانه dynamic را فراخوانی می‌کنند استفاده می‌شود. این دسته از کتابخانه‌ها که عموماً به اسم shared object نیز شناخته می‌شوند، دارای پسوند so. هستند. این کتابخانه‌ها هم از فایل‌های obj ساخته می‌شوند. بنابراین ابتدا باید فایل‌های obj موردنیاز خود را تولید کرد:

```
gcc -c -fPIC dynamicLib.c -o dynamicLib.o
```

استفاده از fPIC باعث می‌شود که کد حاصل مستقل از آدرس و مکان باشد. یعنی از آنجا که برنامه به برنامه‌های مختلف لینک می‌شود نمی‌توان آدرس دقیق سمبل‌ها را تعیین کرد (اصلی‌ترین تفاوت کتابخانه استاتیک و پویا). به همین دلیل با ذکر این flag این محدودیت را نادیده می‌گیریم تا خود سیستم عامل هنگام بارگذاری برنامه اصلی این آدرس‌ها را تعیین کند. حال برای تولید کتابخانه پویا به شکل زیر عمل می‌کنیم:

```
gcc -shared dynamicLib.o -o dynamicLib.so
```

اکنون کتابخانه پویای ما آماده شده‌است. حال فرض کنید که کدی به اسم myapp.c داریم که از توابع موجود در dynamicLib استفاده کرده است. برای اینکه به صورت پویا این کتابخانه را به کد خود لینک کنیم، به شکل زیر عمل می‌کنیم:

```
gcc myapp.c -L./ -ldynamicLib -o app.out
```

با اجرای دستور فوق، فایل اجرایی شما تولید می‌شود. اما شاید هنگام اجرای آن به خطا برخورد کنید و نتوانید آن را اجرا کنید. این خطا احتمالاً به این دلیل است که loader نمی‌تواند کتابخانه پویا را پیدا کند و آن را بارگذاری کند. معمولاً زمانی این مشکل پیش می‌آید که فایل اشتراکی (کتابخانه پویا) در مسیرهای استاندارد قرار ندارد. بنابراین یا کتابخانه پویا را در یک مسیر استاندارد قرار دهید و یا اینکه مسیر موردنظر را به متغیر LD_LIBRARY_PATH اضافه کنید:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH";path_to_shared_object"
Export LD_LIBRARY_PATH
```

4 مثال‌ها

مثال های آورده شده در زیر را نوشته و اجرا کنید:

4.1 open:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>

int main()
{
```

```
	char buffer[256];
	sprintf(buffer,"%s","this is string in open");
	int openFile=open("open.txt",O_CREAT|O_RDWR,00777);
	return 0;
}
```

4.2 write:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main()
{
	char buffer[256];
	sprintf(buffer,"%s","this is string in write");
	int openFile=open("open.txt",O_CREAT | O_TRUNC |
O_RDWR,00777);
	write(openFile,buffer,strlen(buffer));
	return 0;
}
```

4.3 read:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main()
{
	char readBuffer[256];
```

```
	int readFile=open("open.txt",O_RDONLY,00777);
	read(readFile,readBuffer,255);
	fprintf(stdout,"%s\n",readBuffer);
	return 0;
}
```

4.4 mkdir:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
	char dirName[256];
	sprintf(dirName,"%s","/home/dirTest");
	mkdir(dirName,00755);
	return 0;
}
```