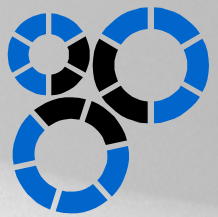
System Programming
Lecture 2

Jubayer Al Mahmud
Lecturer
Jashore University of Science and Technology

Outline

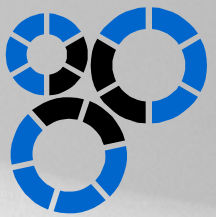
- Linux File APIs
  - creat
  - read
  - write
  - close
  - fcntl
  - lseek
  - link, unlink
  - stat, fstat
  - chmod, fchmod



Linux File API - creat

```
#include <sys/types.h>
#include <unistd.h>
int creat (const char* path_name, mode_t mode);
```

- The ***path_name*** argument is the name of a file to be created.
- The ***mode_t*** argument is to set the access permission of the newly created file.
- Returns the file descriptor on success; Returns -1 on failure.
- ***creat*** is now obsolete as another API named ***open*** can be used to create and open a file by setting ***O_CREAT*** flag.
- **Operation: Creates a file for data access.**



creat()

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


int main() {

mode_t mode = S_IRUSR | S_IWUSR;
creat("test.txt", mode);

return 0;
}
```

Linux File API - read

```
#include <sys/types.h>
#include <unistd.h>
ssize_t read (int fdesc, void* buf, size_t size);
```

- ***fdesc*** is an integer file descriptor that refers to an opened file.
- ***buf*** is the address of a buffer holding any data read
- ***size*** specifies how many bytes of data to be read from the file.
- Returns the number of bytes of data successfully read and stored in the ***buf*** argument. Returns -1 or signal interrupt on failure.
- **Operation: Reads data from a file**

read()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFFER_SIZE 1024

int main() {
int fd;
ssize_t bytes_read;
char buffer[BUFFER_SIZE];

fd = open("test.txt", O_RDONLY);

if (fd == -1) {
perror("open");
_exit(EXIT_FAILURE);
}

bytes_read = read(fd, buffer, BUFFER_SIZE);

if (bytes_read == -1) {
perror("read");
close(fd);
_exit(EXIT_FAILURE);
}

write(STDOUT_FILENO, buffer, bytes_read);

close(fd);

return 0;
}
```

Linux File API - write

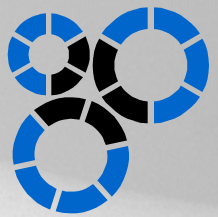
```
#include <sys/types.h>
#include <unistd.h>
ssize_t write (int fdesc, const void* buf, size_t size);
```

- ***fdesc*** is an integer file descriptor that refers to an opened file.
- ***buf*** is the address of a buffer which contains data to be written to the file.
- ***size*** specifies how many bytes of data are in the ***buf*** argument.
- Returns the number of bytes of data successfully written to a file. Returns -1 or signal interrupt on failure.
- **Operation: Writes data to a file**

Linux File API - close

```
#include <unistd.h>
int close (int fdesc);
```

- ***fdesc*** is an integer file descriptor that refers to an opened file.
- Returns 0 on success; returns -1 on failure.
- **Operation: Terminates the connection to a file.**

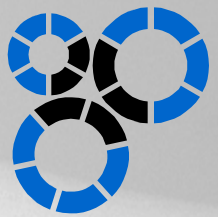


Linux File API - fcntl

```
#include <fcntl.h>
int fcntl (int fdesc, int cmd, ...);
```

- ***cmd*** argument specifies which operation to perform on a file referenced by the ***fdesc*** argument.
- The 3rd argument depends on the value of cmd.
- Possible ***cmd*** values are:

1. ***F_GETFL*** – Returns the access control flags of a file descriptor.
2. ***F_SETFL*** – Sets/clears access control flags that are specified in the 3rd arguemnt to ***fcntl***.



Linux File API - fcntl

3. ***F_GETFD*** – Returns the ***close-on-exec*** flag of a file descriptor ***fdesc***.
4. ***F_SETFD*** – Sets/clears the ***close-on-exec*** flag of a file descriptor ***fdesc***.
5. ***F_DUPFD*** – Duplicates the file descriptor ***fdesc*** with another file descriptor

- ***close-on-exec:*** This flag specifies that if the process that owns the file descriptor calls the exec API to execute a different program, the file descriptor should be closed by the kernel before the new program runs (if the flag is on) or not (if the flag is off).
- **Operation: Helps user to set access control flags and the close-on-exec flag of any file descriptor.**

Linux File API - fcntl

- The ***dup*** & ***dup2*** can be used instead of fcntl

We can use

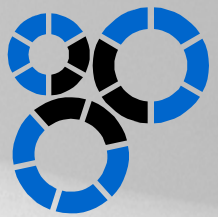
`#define dup(fdesc);` instead of

`fcntl (fdesc, F_DUPFD, 0);`

We can use

`#define dup2 (fdesc1, fd2);` instead of

`fcntl (fdesc, F_DUPFD,fd2);`



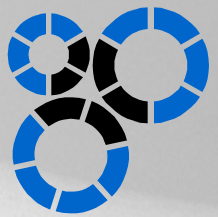
Linux File API – lseek

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek (int fdesc, off_t pos, int whence);
```

- ***pos*** specifies a byte offset to be added to a reference location for finding the new file offset value.
- ***whence*** specifies the reference location.
- ***whence*** can have 3 values:

1. ***SEEK_CUR*** – Reference location is current file pointer address
2. ***SEEK_SET –*** Reference location is the beginning of a file
3. ***SEEK_END –*** Reference location is the end of a file.

- Returns new file offset on success; -1 on failure.
- **Operation: Allows random access of data in a file.**

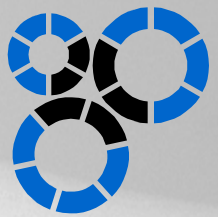


Linux File API - link

```
#include <unistd.h>
int link (const char* cur_link, const char* new_link);
```

- ***cur_link*** is a path name of an existing file.
- ***new_link*** is a path name to be assigned to the same file.
- Increases the hard link count by 1 on success.

- **Operation: Creates a hard link to a file (the process must have super user privilege).**

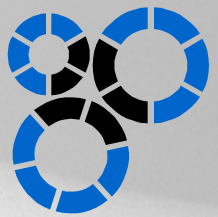


Linux File API - unlink

```
#include <unistd.h>
int unlink (const char* cur_link);
```

- ***cur_link*** is a path name of an existing file.
- Returns 0 on success; returns -1 on failure.

- **Operation: Deletes a hard link of a file (the process must have super user privilege).**



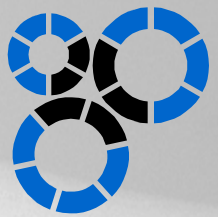
Linux File API – stat, fstat

```
#include <sys/stat.h>
#include <unistd.h>
int stat (const char* path_name,struct stat* statv);
int fstat (const int fdesc,struct stat* statv);
```

- statv argument has 11 values:

```
struct stat
{
dev_ts    t_dev      /*file system ID*/
uid_t     st_uid     /*File user ID*/
time_t    st_atime       /*Last access time*/
time_t    st_mtime       /*Last modification time*/
time_t    st_ctime       /*Last status change time */…
}
```

Operation: Queries attributes of a file



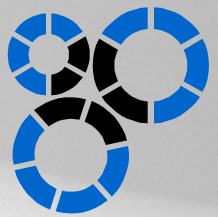
Linux File API – chmod, fchmod

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int chmod (const char* path_name,mode_t flag);
int fchmod (int fdesc,mode_t flag);
```

- The ***flag*** argument contains the new access permission

.

- **Operation: Changes access permission of a file.**



Thank You