Programming Projects

CS-370: Operating System

University of Nevada, Las Vegas

SEB 1242

Getting Started!

- DO NOT USE bobby!!!!

Getting Started!

- DO NOT USE bobby!!!!
- You must use cardiac.cs.unlv.edu to work on your assignments.

Getting Started!

- DO NOT USE bobby!!!!
- You must use cardiac.cs.unlv.edu to work on your assignments.
- The assignments will be graded on cardiac.

Getting Started!

- DO NOT USE bobby!!!!
- You must use cardiac.cs.unlv.edu to work on your assignments.
- The assignments will be graded on cardiac.
- Thursday's class(on September 4, 2014) will be held in the lab (B-361).

Getting Started!

- DO NOT USE bobby!!!!
- You must use cardiac.cs.unlv.edu to work on your assignments.
- The assignments will be graded on cardiac.
- Thursday's class(on September 4, 2014) will be held in the lab (B-361).
- There will be an assignment that must be completed in class, so you must show up in the lab on Thursday.

Getting Started!

- DO NOT USE bobby!!!!
- You must use cardiac.cs.unlv.edu to work on your assignments.
- The assignments will be graded on cardiac.
- Thursday's class(on September 4, 2014) will be held in the lab (B-361).
- There will be an assignment that must be completed in class, so you must show up in the lab on Thursday.
- You must have a CS account and registered on the class web site before you come to class on Thursday.

Projects this semester!

- Programs will have to be written in either C or C++.
- Use gcc to compile the C source.

Projects this semester!

- Programs will have to be written in either C or C++.
- Use gcc to compile the C source.
- All assignments must be turned in on the class website http://osserver.cs.unlv.edu/moodle/

Projects this semester!

- Programs will have to be written in either C or C++.
- Use gcc to compile the C source.
- All assignments must be turned in on the class website http://osserver.cs.unlv.edu/moodle/
- Enrollment key for this class is : osclass (all lower case, one word)

Shell programming project

CS-370: Operating System

University of Nevada, Las Vegas

SEB 1242

System Calls

Mechanism used by a program to request services from operating system.

- Kind of services:
  - hardware related
  - process control related
  - communication related

System Calls

Mechanism used by a program to request services from operating system.

- Kind of services:
  - hardware related
  - process control related
  - communication related
- Mechanism:
  - using library that has the wrapper function.

System Calls

Mechanism used by a program to request services from operating system.

- Kind of services:
  - hardware related
  - process control related
  - communication related
- Mechanism:
  - using library that has the wrapper function.
- What does the library do?
  - assigns unique system call number
  - places the arguments in stack / register
  - traps the kernel

IO: printf

- Used for printing on the screen

IO: printf

- Used for printing on the screen
- Prototype:

```
int printf(const char *format, ...);
```

IO: printf

- Used for printing on the screen
- Prototype:

```
int printf(const char *format, ...);
```

- Library :
  stdio.h

IO: printf

- Used for printing on the screen
- Prototype:

```
int printf(const char *format, ...);
```

- Library :
  stdio.h
- Example:

```
int integer = 8;
char* string = "hello";
printf("number[%d] string[%s]\n", integer, string);
```

IO: printf

- Used for printing on the screen
- Prototype:

```
int printf(const char *format, ...);
```

- Library :
  stdio.h
- Example:

```
int integer = 8;
char* string = "hello";
printf("number[%d] string[%s]\n", integer, string);
```

- Output:
  number[8] string[hello]

IO: printf

- Used for printing on the screen
- Prototype:

```
int printf(const char *format, ...);
```

- Library :
  stdio.h
- Example:

```
int integer = 8;
char* string = "hello";
printf("number[%d] string[%s]\n", integer, string);
```

- Output:
  number[8] string[hello]
- Related: scanf for reading input

IO: perror

- Used to print errors to standard error

IO: perror

- Used to print errors to standard error
- Prototype:

```
void perror(const char *s);
```

IO: perror

- Used to print errors to standard error
- Prototype:

```
void perror(const char *s);
```

- Library :
  stdio.h

IO: perror

- Used to print errors to standard error
- Prototype:

```
void perror(const char *s);
```

- Library :
  stdio.h
- Example:

```
perror(NULL);
```

IO: perror

- Used to print errors to standard error
- Prototype:

```
void perror(const char *s);
```

- Library :
  stdio.h
- Example:

```
perror(NULL);
```

- Note: Input parameter can be null.

IO: fgets

- Used for reading a line of input

IO: fgets

- Used for reading a line of input
- Prototype:

```
char *fgets(char *s, int size, FILE *stream);
*s = buffer to store the line
size = the maximum length to read in (size of *s)
*stream = where to input from (use stdin)
```

IO: fgets

- Used for reading a line of input
- Prototype:

```
char *fgets(char *s, int size, FILE *stream);
*s = buffer to store the line
size = the maximum length to read in (size of *s)
*stream = where to input from (use stdin)
```

- Library :
  stdio.h

IO: fgets

- Used for reading a line of input
- Prototype:

```
char *fgets(char *s, int size, FILE *stream);
*s = buffer to store the line
size = the maximum length to read in (size of *s)
*stream = where to input from (use stdin)
```

- Library :
  stdio.h
- Example:

```
char *buffer = (char *)calloc(256, sizeof(char));
fgets(buffer, 256, stdin);
```

Dynamic Arrays: calloc

- Used to allocate memory

Dynamic Arrays: calloc

- Used to allocate memory
- Prototype:

```
void *calloc(size_t nmemb, size_t size);
nmemb = the number of elements to allocate
size = the size of each element
```

Dynamic Arrays: calloc

- Used to allocate memory
- Prototype:

```
void *calloc(size_t nmemb, size_t size);
nmemb = the number of elements to allocate
size = the size of each element
```

- Library :
  stdlib.h

Dynamic Arrays: calloc

- Used to allocate memory
- Prototype:

```
void *calloc(size_t nmemb, size_t size);
nmemb = the number of elements to allocate
size = the size of each element
```

- Library :
  stdlib.h
- Example:

```
int *array = (int *)calloc(10, sizeof(int));
```

Dynamic Arrays: calloc

- Used to allocate memory
- Prototype:

```
void *calloc(size_t nmemb, size_t size);
nmemb = the number of elements to allocate
size = the size of each element
```

- Library :
  stdlib.h
- Example:

```
int *array = (int *)calloc(10, sizeof(int));
```

- Note : Returns a pointer to memory allocated

Dynamic Arrays: free

- Used to free dynamically allocated memory

Dynamic Arrays: free

- Used to free dynamically allocated memory
- Prototype:

```
void free(void *ptr);
*ptr = pointer to the memory to be freed
```

Dynamic Arrays: free

- Used to free dynamically allocated memory
- Prototype:

```
void free(void *ptr);
*ptr = pointer to the memory to be freed
```

- Library :
  stdlib.h

Dynamic Arrays: free

- Used to free dynamically allocated memory
- Prototype:

```
void free(void *ptr);
*ptr = pointer to the memory to be freed
```

- Library :
  stdlib.h
- Example:

```
int *array = (int *)calloc(10, sizeof(int));
free(array);
```

String Library: strtok

- Used to tokenize a string (char *)

String Library: strtok

- Used to tokenize a string (char *)
- Prototype:

```
char *strtok(char *str, const char *delim);
   *str = string to tokenize
*delim = list of delimiters (characters)
```

String Library: strtok

- Used to tokenize a string (char *)
- Prototype:

```
char *strtok(char *str, const char *delim);
   *str = string to tokenize
*delim = list of delimiters (characters)
```

- Library :
  string.h

String Library: strtok...

- Example:

```
char *string = ``tokenize\n this\n string\n please\n'';
char *token = strtok(string, `` \n'');
while (token != NULL)
{
printf(``%s\n'', token);
token = strtok(NULL, `` \n'');
}
```

String Library: strtok...

- Example:

```
char *string = ``tokenize\n this\n string\n please\n'';
char *token = strtok(string, `` \n'');
while (token != NULL)
{
printf(``%s\n'', token);
token = strtok(NULL, `` \n'');
}
```

- Output:

```
tokenize
this
string
please
```

Process Control: fork

- Used to create and execute a child process

Process Control: fork

- Used to create and execute a child process
- Prototype:

```
pid_t fork(void);
```

Process Control: fork

- Used to create and execute a child process
- Prototype:

```
pid_t fork(void);
```

- Library :
  unistd.h

Process Control: fork

- Used to create and execute a child process
- Prototype:

```
pid_t fork(void);
```

- Library :
  unistd.h
- Returns :
  0 to the child process
  childs pid to the parent

Process Control: fork

- Used to create and execute a child process
- Prototype:

```
pid_t fork(void);
```

- Library :
  unistd.h
- Returns :
  0 to the child process
  childs pid to the parent
- Note:
  After this system call is made there will be two processes executing, the child and parent. The new child process will get it's own address space, initialized from the parent's (copy). There is no shared memory between the parent and child processes. After a child process has completed it becomes a zombie until the parent process cleans it up.

Process Control: fork...

- Example1:

```
unsigned int pid = fork();
printf(``Here\n'');
```

Process Control: fork...

- Example1:

```
unsigned int pid = fork();
printf(``Here\n'');
```

- Output:

```
Here
Here
```

Process Control: fork...

- Example2:

```
unsigned int pid = fork();
If (pid == 0)
{
printf(``Child\n'');
}
  else if (pid >0)
{
printf(``Parent\n'');
}
```

Process Control: fork...

- Example2:

```
unsigned int pid = fork();
If (pid == 0)
{
printf(``Child\n'');
}
  else if (pid >0)
{
printf(``Parent\n'');
}
```

- Output:

```
Child
Parent
```

Or

```
Parent
Child
```

Process Control: waitpid

- Used to wait for a child process to complete

Process Control: waitpid

- Used to wait for a child process to complete
- Prototype:

```
pid_t waitpid(pid_t pid, int *status, int options);
pid = pid of child to wait for, can be -1 meaning wait
    for any child process to complete.
*status = status flag returned containing data on how
    the child ended
options = what kind of wait
WNOHANG for non blocking
WUNTRACED for blocking
```

Process Control: waitpid

- Used to wait for a child process to complete
- Prototype:

```
pid_t waitpid(pid_t pid, int *status, int options);
pid = pid of child to wait for, can be -1 meaning wait
    for any child process to complete.
*status = status flag returned containing data on how
    the child ended
options = what kind of wait
WNOHANG for non blocking
WUNTRACED for blocking
```

- Library :
  sys/types.h, sys/wait.h

Process Control: waitpid

- Used to wait for a child process to complete
- Prototype:

```
pid_t waitpid(pid_t pid, int *status, int options);
pid = pid of child to wait for, can be -1 meaning wait
    for any child process to complete.
*status = status flag returned containing data on how
    the child ended
options = what kind of wait
WNOHANG for non blocking
WUNTRACED for blocking
```

- Library :

  sys/types.h, sys/wait.h
- Returns :

  On success, the pid of the child process that ended

  -1 on error

  0 in the case of WNOHANG and no child process had ended

Process Control: waitpid...

- Example:

```
unsigned int pid = fork();
int status;
if (pid == 0) {
printf(``Child\n'');
exit(0);
} else {
waitpid(pid, &status, WUNTRACED);
printf(``Parent\n'');
}
```

Process Control: waitpid...

- Example:

```
unsigned int pid = fork();
int status;
if (pid == 0) {
printf(``Child\n'');
exit(0);
} else {
waitpid(pid, &status, WUNTRACED);
printf(``Parent\n'');
}
```

- Output:

```
Child
Parent
```

Process Control: execvp

- Used to execute another program from within a process

Process Control: execvp

- Used to execute another program from within a process
- Prototype:

```
int execvp(const char *file, char *const argv[]);
*file = the name of the file (program) to execute
argv[] = an array of arguments for the program.
The first element must contain *file
The last element must be NULL
```

Process Control: execvp

- Used to execute another program from within a process
- Prototype:

```
int execvp(const char *file, char *const argv[]);
*file = the name of the file (program) to execute
argv[] = an array of arguments for the program.
The first element must contain *file
The last element must be NULL
```

- Library :
  unistd.h

Process Control: execvp

- Used to execute another program from within a process
- Prototype:

```
int execvp(const char *file, char *const argv[]);
*file = the name of the file (program) to execute
argv[] = an array of arguments for the program.
The first element must contain *file
The last element must be NULL
```

- Library :
  unistd.h
- Returns :
  -1 on error

Process Control: execvp

- Used to execute another program from within a process
- Prototype:

```
int execvp(const char *file, char *const argv[]);
*file = the name of the file (program) to execute
argv[] = an array of arguments for the program.
The first element must contain *file
The last element must be NULL
```

- Library :
  unistd.h
- Returns :
  -1 on error
- Note:
  If this call executes properly it will end the current process (which makes the process a zombie). If the call did not execute properly the process will stay alive.

Process Control: execvp...

- Example1:

```
char * command[10];
command[0] = "df";
command[1] = "-h";
command[2] = NULL;
pid = fork();
if (pid == 0)
 {
printf("child[%d]\n", getpid());
execvp(command[0], command);
} else { . . . ...}
```

IO Control: dup

- Stands for duplicate, used to duplicate file descriptors.

IO Control: dup

- Stands for duplicate, used to duplicate file descriptors.
- Prototype:

```
int dup(int oldfd);
oldfd = the file descriptor to copy
```

IO Control: dup

- Stands for duplicate, used to duplicate file descriptors.
- Prototype:

```
int dup(int oldfd);
oldfd = the file descriptor to copy
```

- Library :
  unistd.h

IO Control: dup

- Stands for duplicate, used to duplicate file descriptors.
- Prototype:

```
int dup(int oldfd);
oldfd = the file descriptor to copy
```

- Library :
  unistd.h
- Returns:
  the file descriptor it was copied to

IO Control: dup

- Stands for duplicate, used to duplicate file descriptors.
- Prototype:

```
int dup(int oldfd);
oldfd = the file descriptor to copy
```

- Library :
  unistd.h
- Returns:
  the file descriptor it was copied to
- File descriptor standards:
  0 = stdin
  1 = stdout
  2 = stderror

IO Control: close

- Closes a file descriptor

IO Control: close

- Closes a file descriptor
- Prototype:

```
int close(int fd);
fd = file descriptor to close
```

IO Control: close

- Closes a file descriptor
- Prototype:

```
int close(int fd);
fd = file descriptor to close
```

- Library :
  unistd.h

IO Control: close

- Closes a file descriptor
- Prototype:

```
int close(int fd);
fd = file descriptor to close
```

- Library :
  unistd.h
- Example:

```
close(0);
```

IO Control: close

- Closes a file descriptor
- Prototype:

```
int close(int fd);
fd = file descriptor to close
```

- Library :
  unistd.h
- Example:

```
close(0);
```

- Returns:
  0 on success
  -1 on error

Process Communication: pipe

- used to communicate (send data) from one process to another. (only for parent-child communication)

Process Communication: pipe

- used to communicate (send data) from one process to another. (only for parent-child communication)
- Prototype:

```
int pipe(int filedes[2]);
filedes[2] = array containing the descriptors to use
    for communiction
[0] = input
[1] = output
```

Process Communication: pipe

- used to communicate (send data) from one process to another. (only for parent-child communication)
- Prototype:

```
int pipe(int filedes[2]);
filedes[2] = array containing the descriptors to use
    for communiction
[0] = input
[1] = output
```

- Library :
  unistd.h

Process Communication: pipe

- used to communicate (send data) from one process to another. (only for parent-child communication)
- Prototype:

```
int pipe(int filedes[2]);
filedes[2] = array containing the descriptors to use
    for communiction
[0] = input
[1] = output
```

- Library :
  unistd.h
- Returns: 0 on success
  -1 on error

Process Communication: pipe

- used to communicate (send data) from one process to another. (only for parent-child communication)
- Prototype:

```
int pipe(int filedes[2]);
filedes[2] = array containing the descriptors to use
    for communiction
[0] = input
[1] = output
```

- Library :
  unistd.h
- Returns: 0 on success
  -1 on error
- Note:
  Only for unidirectional Communication

Process Communication: pipe...

- Example:

```
char  *str,buf[80];
int fd[2];
pipe(fd);
pid=fork();
if(pid ==0)
{
str ="Hello! This is from child\n";
close(fd[0]);
write(fd[1],str,(strlen(str))+1);
exit(0);
}
else if(pid >0)
{
close(fd[1]);
read(fd[0],buf,sizeof(buf));
printf("%s",buf);
}
```

Signal Handling: signal

- Signal Handling: signal

Signal Handling: signal

- Signal Handling: signal
- Prototype:

```
sighandler_t signal(int signum, sighandler_t handler);
signum = which signal to register a handle for
SIGTSTP = Ctrl-Z
SIGCHLD = when a child process ends
handler = what to do with the signal
Name of a function to be called
+SIG_IGN = ignore the signal (nothing will happen)
+SIG_DFL = default (how ever the signal is handled
    normally)
```

Signal Handling: signal

- Signal Handling: signal
- Prototype:

```
sighandler_t signal(int signum, sighandler_t handler);
signum = which signal to register a handle for
SIGTSTP = Ctrl-Z
SIGCHLD = when a child process ends
handler = what to do with the signal
Name of a function to be called
+SIG_IGN = ignore the signal (nothing will happen)
+SIG_DFL = default (how ever the signal is handled
    normally)
```

- Library :
  signal.h

Signal Handling: signal..

- Example:

```
void handleSignal() {
// signal handeling code ^^ef^^bf^^a2^^ef^^be^^80^^ef^^
    be^^a6
}
int main() {
signal(SIGTSTP, handleSignal);
//^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^a6
}
```

Directory Control: getcwd

- Used to retrieve the pathname of the current working directory

Directory Control: getcwd

- Used to retrieve the pathname of the current working directory
- Prototype:

```
char *getcwd(char *buf, unsigned long size);
*buf = the buffer to store the result
size = the size of the buffer
```

Directory Control: getcwd

- Used to retrieve the pathname of the current working directory
- Prototype:

```
char *getcwd(char *buf, unsigned long size);
*buf = the buffer to store the result
size = the size of the buffer
```

- Library :
  unistd.h

Directory Control: getcwd

- Used to retrieve the pathname of the current working directory
- Prototype:

```
char *getcwd(char *buf, unsigned long size);
*buf = the buffer to store the result
size = the size of the buffer
```

- Library :

  unistd.h
- Example:

```
char *buf = (char *) malloc(100);
getcwd(buf,100);
printf("Current working directory is : %s ",buf);
free(buf);
```

Directory Control: getcwd

- Used to retrieve the pathname of the current working directory
- Prototype:

```
char *getcwd(char *buf, unsigned long size);
*buf = the buffer to store the result
size = the size of the buffer
```

- Library :

  unistd.h
- Example:

```
char *buf = (char *) malloc(100);
getcwd(buf,100);
printf("Current working directory is : %s ",buf);
free(buf);
```

- Returns:

  On error it returns a null pointer and set errno to indicate the error.

  Returns the buf argument on success.

Directory Control: chdir

- Used to change the current directory

Directory Control: chdir

- Used to change the current directory
- Prototype:

```
int chdir(const char *path);
*path = the directory to change to
```

Directory Control: chdir

- Used to change the current directory
- Prototype:

```
int chdir(const char *path);
*path = the directory to change to
```

- Library :
  unistd.h

Directory Control: chdir

- Used to change the current directory
- Prototype:

```
int chdir(const char *path);
*path = the directory to change to
```

- Library :
  unistd.h
- Example:

```
chdir("/home/abc/");
```

Directory Control: chdir

- Used to change the current directory
- Prototype:

```
int chdir(const char *path);
*path = the directory to change to
```

- Library :
  unistd.h
- Example:

```
chdir("/home/abc/");
```

Directory Control: chdir

- Used to change the current directory
- Prototype:

```
int chdir(const char *path);
*path = the directory to change to
```

- Library :
  unistd.h
- Example:

```
chdir("/home/abc/");
```

- R:eturns 0 on success
  -1 on error
- Note: Use perror() to get the error message

Struct: termios

- Library:
  termios.h

Struct: termios

- Library:
  termios.h
- Elements:
  - tcflag_t c_iflag Input Modes.
  - tcflag_t c_oflag Output modes.
  - tcflag_t c_cflag Control modes.
  - tcflag_t c_lflag Local modes.
  - cc_t c_cc[NCCS] Control characters.

Struct: termios...

- Flags:
  - c_lflag
    - ICANON canonical input(enables special characters)
    - ECHO Whether to output every character to the screen as it is typed (we will turn this off and manually output characters)
    - Example:

```
c_lflag &= ~(ICANON|ECHO);
```

  - c_cc
    - V_MIN - The minimum amount of characters to read
    - V_TIME The amount of time between returns (tenths of a second)
    - Example:

```
c_cc[VMIN] = 10;
c_cc[VTIME] = 1;
```

termios: tcgetattr

- Retrieves the current termios configuration and stores it

termios: tcgetattr

- Retrieves the current termios configuration and stores it
- Prototype:

```
int tcgetattr(int fd, struct termios *termios_p);
```

Use 0 for fd stdin

termios: tcgetattr

- Retrieves the current termios configuration and stores it
- Prototype:

```
int tcgetattr(int fd, struct termios *termios_p);
```

  Use 0 for fd stdin
- Library :
  termios.h

termios: tcgetattr

- Retrieves the current termios configuration and stores it
- Prototype:

```
int tcgetattr(int fd, struct termios *termios_p);
```

  Use 0 for fd stdin
- Library :
  termios.h
- Returns: 0 on success
  -1 on error

termios: tcsetattr

- Sets a new configuration to be used for IO

termios: tcsetattr

- Sets a new configuration to be used for IO
- Prototype:

```
int tcsetattr(int fd, int optional_actions, const
    struct termios *termios_p);
Fd - the file descriptor to set the configuration for
    (use 0 for STDIN).
Optional_action ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93
    use TCSANOW to make the configuration take affect
    now
```

termios: tcsetattr

- Sets a new configuration to be used for IO
- Prototype:

```
int tcsetattr(int fd, int optional_actions, const
    struct termios *termios_p);
Fd - the file descriptor to set the configuration for
    (use 0 for STDIN).
Optional_action ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93
    use TCSANOW to make the configuration take affect
    now
```

- Library :
  termios.h

termios: tcsetattr

- Sets a new configuration to be used for IO
- Prototype:

```
int tcsetattr(int fd, int optional_actions, const
    struct termios *termios_p);
Fd - the file descriptor to set the configuration for
    (use 0 for STDIN).
Optional_action ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93
    use TCSANOW to make the configuration take affect
    now
```

- Library :
  termios.h
- Returns: 0 on success
  -1 on error

termios....

- Example

```
// get the original configuration
struct termios origConfig;
tcgetattr(0, &origConfig);

// create a copy of the original configuration
struct termios newConfig = origConfig;

// adjust the new configuration
newConfig.c_lflag &= ~(ICANON|ECHO);
newConfig.c_cc[VMIN] = 10;
newConfig.c_cc[VTIME] = 2;

// set the new configuration
tcsetattr(0, TCSANOW, &newConfig);

// restore the original configuration when done
tcsetattr(0, TCSANOW, &origConfig);
```

read

- Used for reading bytes of data into the buffer

read

- Used for reading bytes of data into the buffer
- Prototype:

```
ssize_t read(int fd, void *buf, size_t count);
Fd ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 file descriptor
     to read from
*buf the buffer to store the read bytes to
Count ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 the maximum
    number of bytes to read
```

read

- Used for reading bytes of data into the buffer
- Prototype:

```
ssize_t read(int fd, void *buf, size_t count);
Fd ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 file descriptor
     to read from
*buf the buffer to store the read bytes to
Count ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 the maximum
    number of bytes to read
```

- Library :
  unistd.h

read

- Used for reading bytes of data into the buffer
- Prototype:

```
ssize_t read(int fd, void *buf, size_t count);
Fd ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 file descriptor
     to read from
*buf the buffer to store the read bytes to
Count ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 the maximum
    number of bytes to read
```

- Library :

  unistd.h
- Example:

```
char *buf = (char *) calloc(256, sizeof(char));
int bytesRead;
bytesRead = read(0, buf, 256);
```

read

- Used for reading bytes of data into the buffer
- Prototype:

```
ssize_t read(int fd, void *buf, size_t count);
Fd ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 file descriptor
     to read from
*buf the buffer to store the read bytes to
Count ^^ef^^bf^^a2^^ef^^be^^80^^ef^^be^^93 the maximum
    number of bytes to read
```

- Library :

  unistd.h
- Example:

```
char *buf = (char *) calloc(256, sizeof(char));
int bytesRead;
bytesRead = read(0, buf, 256);
```

- Returns:

  The number of bytes.

Copy-On-Write

- Used for Optimization

Copy-On-Write

- Used for Optimization
- Even after fork, child is given the pointer to the parent's resources till one of them modifies.