Assignment 1 – Study of System Calls and System Commands

Date: 07/03/2022

1. Observation – Study of System calls and System commands

SYSTEM COMMANDS:

cp: Command used to copy files or directories from a source to a destination

Options:

-i : Used to prompt before overwrite (overrides a previous -n option).

-r : Recursive - option used to copy directories recursively.

-s : Symbolic link - option used to make symbolic links.

Syntax: cp <option> <source(s)> <destination>

Example: cp -r dir1 dir2

cp assgn.txt dir2

mv: Command used to move (or) rename files

Options:

-i : Interactive - option used to prompt before overwriting a file.

-b : Backup - option used to make a backup of each existing destination file, but doesn't accept arguments.

-f : Used to remove prompting before overwriting.

Syntax: mv <option> <source> <destination>

mv <oldname> <newname>

Example: mv ex1.txt assgn.txt

mv assgn.txt ../Desktop

ls: Command used to list information about the contents of a directory

Options:

-l : Long list order: used to display directory contents in long list order.

-R : Recursive - used to display the subdirectories recursively.

-r : Reverse - used to display the contents of the directory in reverse order.

Syntax: ls <option> <directory>

Example: ls -R Desktop

ls -l Downloads/dir1

grep: Command used to print the lines matching a given pattern from a file

If no files are specified, grep searches the standard input.

Options:

-v : Invert match - used to print non-matching lines.

-n : Line number - used to print the matching lines with each line prefixed by its line number from the file.

-c : Count - used to suppress the normal output and print the count of matching lines from the input file.

Syntax: grep <option> 'pattern' <filename>

Example: grep -n "System" assgn.txt

chmod: Command used to change the permissions for a file or directory by changing the filemode bits

Options:

-f :Suppress the error messages.

-R : used to change files and directories recursively.

-v : output a diagnostic for every file processed.

Syntax: chmod <option> MODE <file/directory>

Example: chmod u=rwx assgn.txt   //using symbolic code

       chmod 755 assgn.txt   //using octal code

cat: Command used to concatenate files and print on standard output

When no file is given, it reads standard output.

Options:

-n : used to number all the output lines.

-T : used to display tab characters as '^I'.

-s : used to suppress repeated empty output lines.

Syntax: cat <option> <file>

Example: cat assgn.txt

mkdir: Command used to create a new directory

Options:

-m : used to set the file mode of the new directory.

-p : used to make parent directories as needed.

-v : used to print a message for each created directory.

Syntax: mkdir <option> <directoryname>

Example: mkdir newdir

rm: Command used to remove files or directories

By default, it does not remove directories.

Options:

-i : used to prompt before removing a file or directory.

-d : used to remove empty directories.

-r : used to remove directories and their contents recursively.

Syntax: rm <option> <file/directory>

Example: rm -d newdir

rmdir: Command used to remove empty directories

Options:

-p : Parents - used to remove a given directory including its parent directories.

-v : output a diagnostic for every directory processed.

Syntax: rmdir <option> <directory>

Example: rmdir newdir

wc: Command used to print the count of the number of lines, words and characters in a file

Options:

-c : used to print the number of characters in a file.

-w : used to print the number of words in a file.

-l : used to print the number of lines in the file.

Syntax: wc <option> <file>

Example: wc -w assgn.txt

who: Command used to show which user is currently logged into the system

Options:

-q : used to print all login names and number of users logged in.

-m : used to print only hostname and the user associated with stdin.

-r : used to print current run level.

Syntax: who <option> <file/arguments>

Example: who -q

head: Command used to output the first n lines of a file to the standard output

When no file is given, it reads the standard output.

Options:

-n : prints the first n lines from the top of a file.

-v : used to always print headers giving filenames.

-c : prints the first n bytes from the beginning of the file.

Syntax: head <option> <filename>

Example: head -6 assgn.txt

tail: Command used to output the last n lines of a file to standard output

When no file is given, it reads standard output.

Options:

-n : prints the last n lines from the file.

-c : prints the last n bytes from the file.

-v : used to always output headers giving filenames.

Syntax: tail <option> <filename>

Example: tail -4 assgn.txt

nl: Command used to write each file to standard output, with line numbers added

When no file is given, it reads from standard input.

Options:

-p : do not reset line numbers at logical pages.

-n: used to insert line numbers according to a given format.

-v : first line number on each logical page.

Syntax: nl <option> <file>

Example: nl -p assgn.txt

awk: A pattern scanning and text processing language consisting of a sequence of pattern-action pairs acting on a file

Options:

-F : used to set the field separator to a given value.

-f : used to read the program text from a file instead of the command line.

-v : used to assign a value to a program variable.

Syntax: awk <options> 'Program text' <filename>

Example: awk '{print $0}' assgn.txt

SYSTEM CALLS:

fork():

This function is used to create a new process by duplicating the existing process from which it is called. The existing process from which this function is called is called parent process and the newly created process is called child process.

Header file: unistd.h

Syntax: pid_t fork(void);

Arguments: None

Return type: On success, PID is returned in the parent and 0 is returned in the child. On failure, -1 is returned to the parent.

execl():

This function is used to execute a file which is residing in an active process. It replaces the previous executable file and the new file is executed.

Header file: unistd.h

Syntax: int execl(const char* path, const char* arg……./* (char*) NULL */);

Arguments: Path of the executable binary file and arguments (i.e. -lh/home) to be passed to the executable file followed by NULL.

Return type: Returns -1 if error occurs(failure), else does not return anything(success).

getpid():

This function returns the process id of the calling process. It is often used by routines that generate unique temporary filenames.

Header file: unistd.h

Syntax: pid_t getpid(void);

Arguments: None

Return type: Always returns the process ID of the calling process. It never throws any errors, so it is always successful.

getppid():

This function is used to return the process id of the parent of the calling process.

Header file: unistd.h

Syntax: pid_t getppid(void);

Arguments: None

Return type: Always returns the process ID of the parent of the calling process. It never throws any errors, so it is always successful.

exit():

This function is used to terminate the normal process and return the value of status to the parent.

Header file: stdlib.h

Syntax: void exit(int status);

Arguments: An integer value of the exit status.

Return type: The exit() function does not return anything, either during success or failure.

wait():

This function suspends the execution of the calling process until one of its child processes terminates.

Header file: sys/wait.h

Syntax: pid_t wait(int *status);

Arguments: Address of the status(integer value).

Return type: On success, it returns the process id of the terminated child. On failure, it returns -1.

close():

This function is used to close a file descriptor, so that it longer refers to any file and can be reused. Any record locks held on the file it was associated with, and owned by the process, are removed.

Header file: unistd.h

Syntax: int close(int fd);

Arguments: Value of file descriptor 'fd'.

Return type: On success, it returns 0. On failure, it returns -1.

opendir():

This function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

Header file: dirent.h

Syntax: DIR *opendir(const char *name);

Arguments: Name of the directory

Return type: On success, it returns a pointer to the directory stream. On failure, it returns NULL.

readdir():

This function returns a pointer to the dirent structure representing the next directory entry in the directory stream pointed to by dirp.

Header file: dirent.h

Syntax: struct dirent *readdir(DIR *dirp);

Arguments: Directory pointer 'dirp'

Return type: On success, it returns a pointer to a dirent structure. On failure, it returns NULL.

open():

This function returns a file descriptor for the given file pathname, for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest numbered file descriptor not currently open for the process.

Header file: fcntl.h

Syntax: int open(const char *pathname, int flags);

Return type: On success, it returns the file descriptor(integer). On failure, it returns -1.

read():

This function is used to read upto 'count' bytes from a given file descriptor 'fd' into the buffer starting at 'buf'.

Header file: unistd.h

Syntax: ssize_t read(int fd, void *buf, size_t count);

Arguments: A file descriptor, buffer and number of bytes to be read.

Return type: On success, it returns the number of bytes read. On failure, it returns -1.

write():

This function is used to write upto 'count' bytes from the buffer starting at 'buf' to the file referred to by the file descriptor 'fd'.

Header file: unistd.h

Syntax: ssize_t write(int fd, const void *buf, size_t count);

Arguments: File descriptor, buffer and number of bytes to be written.

Return type: On success, it returns the number of bytes written. On failure, it returns -1.

creat():

This function is equivalent to the open() system call. It is actually a redundant function which has been primarily included for historical purposes since many applications depend on it. It creates a new file or rewrites an existing one.

Header file: fcntl.h

Syntax: int creat(const char *pathname, mode_t mode);

Arguments: Pathname of the file, mode

Return type: On success, it returns the file descriptor. On failure, it returns -1.

sleep():

This function is used to delay or pause for a specified amount of time (in seconds).

Header file: unistd.h

Syntax: unsigned int sleep(unsigned int seconds);

Arguments: Delay time in seconds.

Return type: Zero if the requested time has elapsed, or the number of seconds left to sleep, if the call was interrupted by a signal handler

2. Develop a C program to understand the working of fork().

<u>Aim:</u>

To develop a C program to understand the working of fork().

<u>Algorithm:</u>

1. Start
2. Get a process id by forking.
3. If the process id is 0, then proceed to the child process. This will print a chosen unique message a certain number of times.
4. If not, then proceed to the parent process. This will print a different identifying message a given number of times.
5. Stop

<u>Code:</u>

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

#define   MAX_COUNT  200


void  ChildProcess(void);
void  ParentProcess(void);

void  main(void) {
      pid_t  pid;
      pid = fork();
      if (pid == 0)
            ChildProcess();
      else
            ParentProcess();
      }

void  ChildProcess(void) {
      int   i;
      for (i = 1; i <= MAX_COUNT; i++)
            printf("\tThis line is from child, value = %d\n", i);
      printf("\t*** Child process is done ***\n");
      }
```

```
void  ParentProcess(void) {
      int   i;
      for (i = 1; i <= MAX_COUNT; i++)
              printf("This line is from parent, value = %d\n", i);
      printf("*** Parent is done ***\n");
      }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o fork fork_call.c
kri@kri-ubuntu:~/workspace$ ./fork
This line is from parent, value = 1
This line is from parent, value = 2
This line is from parent, value = 3
This line is from parent, value = 4
This line is from parent, value = 5
        This line is from child, value = 1
        This line is from child, value = 2
        This line is from child, value = 3
        This line is from child, value = 4
This line is from parent, value = 6
        This line is from child, value = 5
This line is from parent, value = 7
        This line is from child, value = 6
This line is from parent, value = 8
        This line is from child, value = 7
This line is from parent, value = 9
        This line is from child, value = 8
This line is from parent, value = 10
        This line is from child, value = 9
This line is from parent, value = 11
        This line is from child, value = 10
This line is from parent, value = 12
        This line is from child, value = 11
This line is from parent, value = 13
        This line is from child, value = 12
This line is from parent, value = 14
        This line is from child, value = 13
This line is from parent, value = 15
        This line is from child, value = 14
This line is from parent, value = 16
        This line is from child, value = 15
This line is from parent, value = 17
        This line is from child, value = 16
This line is from parent, value = 18
        This line is from child, value = 17
This line is from parent, value = 19
        This line is from child, value = 18
This line is from parent, value = 20
        This line is from child, value = 19
This line is from parent, value = 21
        This line is from child, value = 20
This line is from parent, value = 22
        This line is from child, value = 21
This line is from parent, value = 23
        This line is from child, value = 22
This line is from parent, value = 24
        This line is from child, value = 23
This line is from parent, value = 25
        This line is from child, value = 24
This line is from parent, value = 26
        This line is from child, value = 25
This line is from parent, value = 27
        This line is from child, value = 26
This line is from parent, value = 28
        This line is from child, value = 27
This line is from parent, value = 29
        This line is from child, value = 28
This line is from parent, value = 30
        This line is from child, value = 29
This line is from parent, value = 31
        This line is from child, value = 30
This line is from parent, value = 32
        This line is from child, value = 31
This line is from parent, value = 33
        This line is from child, value = 32
This line is from parent, value = 34
        This line is from child, value = 33
This line is from parent, value = 35
        This line is from child, value = 34
This line is from parent, value = 36
        This line is from child, value = 35
This line is from parent, value = 37
        This line is from child, value = 36
This line is from parent, value = 38
        This line is from child, value = 37
This line is from parent, value = 39
        This line is from child, value = 38
This line is from parent, value = 40
        This line is from child, value = 39
This line is from parent, value = 41
        This line is from child, value = 40
This line is from parent, value = 42
        This line is from child, value = 41
This line is from parent, value = 43
        This line is from child, value = 42
This line is from parent, value = 44
        This line is from child, value = 43
This line is from parent, value = 45
        This line is from child, value = 44
This line is from parent, value = 46
        This line is from child, value = 45
This line is from parent, value = 47
        This line is from child, value = 46
This line is from parent, value = 48
        This line is from child, value = 47
This line is from parent, value = 49
        This line is from child, value = 48
This line is from parent, value = 50
```

```
        This line is from child, value = 49
This line is from parent, value = 51
        This line is from child, value = 50
This line is from parent, value = 52
        This line is from child, value = 51
This line is from parent, value = 53
        This line is from child, value = 52
This line is from parent, value = 54
        This line is from child, value = 53
This line is from parent, value = 55
        This line is from child, value = 54
This line is from parent, value = 56
        This line is from child, value = 55
This line is from parent, value = 57
        This line is from child, value = 56
This line is from parent, value = 58
        This line is from child, value = 57
This line is from parent, value = 59
        This line is from child, value = 58
This line is from parent, value = 60
        This line is from child, value = 59
This line is from parent, value = 61
        This line is from child, value = 60
This line is from parent, value = 62
        This line is from child, value = 61
This line is from parent, value = 63
        This line is from child, value = 62
This line is from parent, value = 64
        This line is from child, value = 63
This line is from parent, value = 65
        This line is from child, value = 64
This line is from parent, value = 66
        This line is from child, value = 65
This line is from parent, value = 67
        This line is from child, value = 66
This line is from parent, value = 68
        This line is from child, value = 67
This line is from parent, value = 69
```

.
.
.

```
        This line is from child, value = 183
This line is from parent, value = 184
        This line is from child, value = 184
This line is from parent, value = 185
        This line is from child, value = 185
This line is from parent, value = 186
        This line is from child, value = 186
This line is from parent, value = 187
        This line is from child, value = 187
This line is from parent, value = 188
        This line is from child, value = 188
This line is from parent, value = 189
        This line is from child, value = 189
This line is from parent, value = 190
        This line is from child, value = 190
This line is from parent, value = 191
        This line is from child, value = 191
This line is from parent, value = 192
        This line is from child, value = 192
This line is from parent, value = 193
        This line is from child, value = 193
This line is from parent, value = 194
        This line is from child, value = 194
This line is from parent, value = 195
        This line is from child, value = 195
This line is from parent, value = 196
        This line is from child, value = 196
This line is from parent, value = 197
        This line is from child, value = 197
This line is from parent, value = 198
        This line is from child, value = 198
This line is from parent, value = 199
        This line is from child, value = 199
This line is from parent, value = 200
        This line is from child, value = 200
*** Parent is done ***
        *** Child process is done ***
```

3. Develop a C program using system calls to open a file, read the contents of the same, display it and close the file. Use command line arguments to pass the file name to the program.

Aim:

To develop a C program using system calls to open a file, read the contents of the same, display it and close the file.

Algorithm:

1. Start
2. Check if the number of command line arguments is 2. If it is greater than 2, print that there are too many arguments. If it is lesser, then print that an argument is expected.
3. Find the file desciptor for the source file given as the argument. Include an error message for the case when the source file is not found.
4. Save the contents of the source file in a string.
5. Display the contents of the file by printing the string.
6. Close the source file.
7. Stop

Code:

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

#define MAX_LENGTH 100
#define MAX_BYTES 500

int main (int argc, char* argv[]){
        if (argc == 2){
                char src[MAX_LENGTH];
                strcpy(src,argv[1]);
                int src_fd, src_sz;
                char* content = (char *) calloc(100, sizeof(char));
                printf("\n");

                //find file desciptor for src file
                src_fd = open(src,O_RDONLY);
                //include error msg for case when src file not found
                if (src_fd < 0) { perror("Command unsuccessful. Source file not found."); exit(1); }
```

```
                //save the contents of src file in a string
                src_sz = read(src_fd,content,MAX_BYTES);
                content[src_sz] = '\0';

                //display the contents of the file by printing the string
                printf("%s\n",content);

                //close src file
                close(src_fd);
                }

        else if (argc > 2){
                printf("Too many arguments supplied.\n");
                }

        else {
                printf("An argument is expected.\n");
                }
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o readf readf.c
kri@kri-ubuntu:~/workspace$ ./readf
An argument is expected.
kri@kri-ubuntu:~/workspace$ ./readf arg1 arg2
Too many arguments supplied.
kri@kri-ubuntu:~/workspace$ ./readf randfile

Command unsuccessful. Source file not found.: No such file or directory
kri@kri-ubuntu:~/workspace$ ./readf foo.txt

This is the first line.
This is the second line.
.

.

.
This is the last line.

kri@kri-ubuntu:~/workspace$
```

Learning outcomes:

- Various system calls and commads were studied.

- The common options used with each command were observed.

- The syntax and header files required for each command were understood.

- The purpose and applications of each command were studied.