1.PIPES

~$ cat pipes1.c

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

// Writing to external command

int main() {

FILE \*write\_fp;

char buffer[BUFSIZ + 1];

sprintf(buffer, "The quick brown fox is sleeping.\n");

write\_fp = popen("cat > newfile.txt", "w");

if (write\_fp != NULL) {

fwrite(buffer, sizeof(char), strlen(buffer), write\_fp);

pclose(write\_fp);

exit(EXIT\_SUCCESS);

}

exit(EXIT\_FAILURE);

}

~$ cc pipes1.c

~$ ./a.out

~$ cat newfile.txt

The quick brown fox is sleeping.

~$

2.

~$ cat pipes2.c

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

// Communication between child and parent

int main() {

int data\_processed;

int file\_pipes[2];

const char some\_data[] = "Hello World";

char buffer[BUFSIZ + 1];

pid\_t fork\_result;

memset(buffer, '\0', sizeof(buffer));

if (pipe(file\_pipes) == 0) {

fork\_result = fork();

if (fork\_result == -1) {

fprintf(stderr, "Fork failure");

exit(EXIT\_FAILURE);

}

else if (fork\_result == 0) { // Child

sleep(2);

data\_processed = read(file\_pipes[0], buffer, BUFSIZ);

printf("Read %d bytes: %s.\n", data\_processed, buffer);

exit(EXIT\_SUCCESS);

}

else { // Parent

data\_processed = write(file\_pipes[1], some\_data, strlen(some\_data));

printf("Wrote %d bytes.\n", data\_processed);

}

}

exit(EXIT\_SUCCESS);

}

~$ cc pipes2.c

~$ ./a.out

Wrote 11 bytes.

~$ Read 11 bytes: Hello World.

-----------------------------------\_----------------\_--------------------

2. SIGNALS

1.

~$ cat signals1.c

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void oh(int sig) {

printf("OH! I got signal %d\n", sig);

signal(SIGINT, oh);

}

int main() {

signal(SIGINT, oh);

while (1) {

printf("Hello World!\n");

sleep(1);

}

}

~$ cc signals1.c

~$ ./a.out

Hello World!

Hello World!

Hello World!

Hello World!

Hello World!

Hello World!

^COH! I got signal 2

Hello World!

Hello World!

Hello World!

^COH! I got signal 2

Hello World!

Hello World!

Hello World!

^Z

[1]+ Stopped ./a.out

~$

2.

~$ cat signals2.c

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void oh(int sig) {

printf("OH! I got signal %d\n", sig);

signal(SIGINT, SIG\_DFL);

}

int main() {

signal(SIGINT, oh);

while (1) {

printf("Hello World!\n");

sleep(1);

}

}

~$ cc signals2.c

~$ ./a.out

Hello World!

Hello World!

Hello World!

Hello World!

Hello World!

^COH! I got signal 2

Hello World!

Hello World!

Hello World!

Hello World!

^C

~$

3.

~$ cat signals3.c

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void oh(int sig) {

printf("OH! I got signal %d\n", sig);

signal(SIGINT, oh);

}

int main() {

signal(SIGQUIT, oh);

while (1) {

printf("Hello world!\n");

sleep(1);

}

}

~$ cc signals3.c

~$ ./a.out

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

^C

~$

4.

~$ cat signals4.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main(void) {

pid\_t ppid, pid, cpid;

ppid = getpid();

pid = fork();

if (ppid == getpid())

printf("parent");

else if (cpid == getpid())

printf("child");

if (pid > 0) {

int i = 0;

while (i++ < 5) {

printf("In the parent process.%d\n", i);

sleep(1);

}

}

else if (pid == 0) {

int i = 0;

while (i++ < 10) {

printf("In the child process.%d\n", i);

sleep(1);

if (i == 3) {

kill(ppid, SIGKILL);

printf("Parent killed. I'm orphan!!!\n");

}

}

}

else {

printf("Something bad happened.");

exit(EXIT\_FAILURE);

}

return 0;

}

~$ cc signals4.c

~$ ./a.out

parentIn the parent process.1

In the child process.1

In the parent process.2

In the child process.2

In the parent process.3

In the child process.3

In the parent process.4

Parent killed. I'm orphan!!!

In the child process.4

Killed

~$ ^C

~$ In the child process.5

In the child process.6

In the child process.7

^C

~$ In the child process.8

In the child process.9

In the child process.10

^C

~$