// 1.Test whether the process(exec() system call) that replaces old program data , will inherit the fds or not.

#include<stdio.h>

#include<unistd.h>

int main(){

printf("I'm going to print the process tree on terminal\n");

int ret = execl("/home/axay/LINUX-INTERNALS/10-3-2022/prob3", "", 0);

if(ret == -1){

printf("execl returned error %d\n", ret);

}

return 0;

}

//to check that FD is inherited or not…

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main(){

int fd;

fd = open("/home/axay/LINUX-INTERNALS/10-3-2022/prob2", O\_RDONLY, 777);

printf("file descriptor of a open system call is %d\n", fd);

return 0;

}

Output:

Text

Description automatically generated

// 2.Write a program such that parent process create two child processes,such that

// each child executes a separate task.

#include <stdio.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int pid1, pid2, pid3;

printf("\nparent process is:%d\n", getpid());

pid1 = fork();

if (pid1 == 0)

{

printf("\n1st child is:%d\n", getpid());

}

else

{

printf("\n2nd child is:%d\n", getpid());

pid2 = fork();

if (pid2 == 0)

{

// execl("/usr/bin/pstree","pstree",NULL);

printf("successfully run execl() for child process 1\n");

}

else

{

// wait(0);

printf("failed to run execl() for child process 1\n");

}

wait(0);

pid3 = fork();

if (pid3 == 0)

{

// execl("/usr/bin/vim","vim","txt3.txt",NULL);

printf("successfully run execl() for child process 2\n");

}

else

{

// wait(0);

printf("failed to run execl() for child process 2\n");

}

}

exit(0);

return 0;

}

Text

Description automatically generated

// 3. A program that replaces old program with new program data and is expected to display

// the currently running processes in a hierarchical tree format.

#include<stdio.h>

#include<unistd.h>

int main(){

printf("I'm going to display a process tree\n");

// int ret = execl("/usr/bin/vim", "vim","info1.txt", 0);

int ret = execl("/usr/bin/pstree", "pstree", 0, 0);

if(ret == -1){

printf("execl returned error %d\n", ret);

}

return 0;

}

Text, timeline

Description automatically generated

// 4.A processs using execl() system call should replace a new command line program.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(int argc, char\* argv[]){

// int ret = execl("/usr/bin/vim", "vim","info1.txt", 0);

int ret = execl(argv[1], argv[2],argv[3], 0);

if(ret == -1){

printf("execl returned error %d\n", ret);

}

return(0);

}

Timeline

Description automatically generated

// 5(a).Write a program parent process wait untill ,while child process open a file and read file

// data into empty buffer.

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main(){

int pid, fd;

pid = fork();

if(pid==0){

// run this as s child process

fd = open("linux400.txt", O\_CREAT | O\_RDWR, 777);

char to\_write[] = "this lines will be added to file. this one is also be added";

int leng = strlen(to\_write);

char to\_read[leng];

int i;

if (fd>0){

if((i = write(fd, to\_write, leng)) < 0){

perror("error occured while writing to a file");

}

}

read(fd, to\_read, leng);

} else{

// run this as a parent process

wait(0);

printf("parent process\n");

}

return 0;

}

Text

Description automatically generated

// 5(b).Write a program, where functions of the program are called in the reverse order of

// their function calls from main().

#include<stdio.h>

#include<stdlib.h>

void callBack1(){

printf("callback 1\n");

}

void callBack2(){

printf("callback 2\n");

}

void callBack3(){

printf("callback 3\n");

}

int main(){

printf("registering callback1\n");

atexit(callBack1);

printf("registering callback2\n");

atexit(callBack2);

printf("registering callback3\n");

atexit(callBack3);

printf("exit() main exiting now...\n");

exit(0); //\_exit() is not calling the functions which are previously registered using atexit function

}

Text

Description automatically generated

// 6. Write a program child executes(exec()) a new program , while parent waits for

// child task to get complete.

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main(){

int pid;

pid = fork();

if(pid==0){

// run this as s child process

printf("this is a child process\n");

printf("new child Process pid = %d\n", getpid());

printf("new child Parent Process pid = %d\n", getppid());

int ret = execl("/usr/bin/vim", "vim","info1.txt", 0);

// int ret = execl("/bin/ls", "ls", 0);

} else{

// run this as a parent process

wait(0);

printf("this is a parent process\n");

printf("parent Process pid = %d\n", getpid());

printf("child Parent's parent Process pid = %d\n", getppid());

}

while(1);

return 0;

}

Shape

Description automatically generated with medium confidence