Assignment 4

U23AI106 || OS LAB Ass 4

===========================================

1. Perform Following

a) Obtain a listing of processes by using the ps command.

b) Use a Linux command which displays a tree of all processes in the system.

c) Use Exec(), Wait() and Exit() system call.

d) Start a browser and check the process ID of the browser using Top command

e) Kill the browser process using “kill” command

f) Change the priority of any user process by using “nice” commad

2. Write a C program using the fork() system call that generates this sequence in the child process.

The starting number will be provided from the user. For example, if 8 is passed as a parameter

on the command line, the child process will output 8, 4, 2, 1. Because the parent and child

processes have their own copies of the data, it will be necessary for the child to output the

sequence. Parent invoke the wait() call to wait for the child process to complete before exiting

the program. Ensure that a positive integer is passed on the command line.

3. Check output and Reason for it.

4. What happens if parent is killed (status of child), what happens when child is killed, what

happens when terminal is killed (). Try these in different sequences - parent, terminal; child

terminal; terminal parent etc.

5. Process Variables and System Calls

a) Create a child process using fork() and explore the effects of:

exec()

wait()

yield()

sleep()

b) Verify whether local, global, and static variables are shared between parent and child

processes.

c) Verify the parent of a parent process.

Codes=>2)

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

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

if (argc != 2) {

fprintf(stderr, "Usage: %s <positive integer>\n", argv[0]);

return 1;

}

int num = atoi(argv[1]);

if (num <= 0) {

fprintf(stderr, "Please provide a positive integer.\n");

return 1;

}

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

} else if (pid == 0) {

while (num >= 1) {

printf("%d ", num);

num /= 2;

}

printf("\n");

} else {

wait(NULL);

printf("Parent process completed.\n");

}

return 0;

}

3)

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int cnt=0;

int main(){

// int a=10;

// if(fork()==0){

// a++;

// }

// printf("%d\n",a);

int i;

for(i=0;i<10;i++){

if(!(i&1)){

fork();

cnt++;

}

}

printf("%d",cnt);

return 0;

}

4)

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

#include <stdlib.h>

void handle\_signal(int sig) {

if (sig == SIGINT) {

printf("Process %d received SIGINT. Terminating...\n", getpid());

exit(0);

}

}

int main() {

pid\_t pid;

signal(SIGINT, handle\_signal);

pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

}

if (pid == 0) {

printf("Child process: My PID is %d, Parent PID is %d\n", getpid(), getppid());

sleep(5);

printf("Child process: After sleep, exiting...\n");

exit(0);

} else {

printf("Parent process: My PID is %d, Child PID is %d\n", getpid(), pid);

sleep(2);

printf("Parent process: Sending SIGKILL to the child process...\n");

kill(pid, SIGKILL);

wait(NULL);

printf("Parent process: After child termination, exiting...\n");

}

return 0;

}

5) #include<iostream>

#include<unistd.h> // fork

#include<sys/types.h> //pID

#include<sys/wait.h>

#include<cstdlib> //exit()

using namespace std;

int main(){

pid\_t pid=fork();

if(pid<0){

cerr<<"Fork Failed"<<endl;

exit(1);

}

else if(pid==0){

cout<<"Child Proccess(PID:)"<<getpid()<<")"<<endl;

execl("/bin/ls","ls","-l",NULL);

cerr<<"exce failed"<<endl;

exit(1);

}

else{

cout<<"Parent proccess is PID:"<<getpid()<<" waiting for child To execute The proccess"<<endl;

int status;

wait(&status);

if(WIFEXITED(status)){

cout<<"Child proccess exited with status :"<<WIFEXITED(status)<<endl;

}else{

cout<<"Child proccess did not exited Normally"<<endl;

}

cout<<"Prent proccess exited "<<endl;

exit(0);

}

}

5) #include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

}

if (pid == 0) {

printf("Child process: Before exec, My PID is %d\n", getpid());

execlp("/bin/ls", "ls", NULL);

perror("Exec failed");

} else {

printf("Parent process: My PID is %d, Child PID is %d\n", getpid(), pid);

wait(NULL);

printf("Parent process: Child has finished.\n");

printf("Parent process: Yielding CPU...\n");

sched\_yield();

printf("Parent process: Sleeping for 2 seconds...\n");

sleep(2);

printf("Parent process: Exiting.\n");

}

return 0;

}

5) #include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int global\_var = 10;

static int static\_var = 20;

int main() {

pid\_t pid;

int local\_var = 30;

pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

}

if (pid == 0) {

global\_var++;

static\_var++;

local\_var++;

printf("Child process: global\_var = %d, static\_var = %d, local\_var = %d\n", global\_var, static\_var, local\_var);

} else {

wait(NULL);

printf("Parent process: global\_var = %d, static\_var = %d, local\_var = %d\n", global\_var, static\_var, local\_var);

}

return 0;

}

5) #include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid, ppid;

pid = fork();

if (pid == 0) {

ppid = getppid();

printf("Child process: My PID is %d, My Parent PID is %d\n", getpid(), ppid);

ppid = getppid();

printf("Child process: My Grandparent PID is %d\n", ppid);

} else {

printf("Parent process: My PID is %d\n", getpid());

wait(NULL);

}

return 0;

}

