**Using fork() and Exec() and process control**

**Question 1: Write a C program that creates two children using the fork().**

**Each process should print its own PID and its PPID. Furthermore, the parent**

**process should print the PIDs of its two child processes.**

**Important: the child processes should not create any new children.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(void){

pid\_t pid1, pid2;

printf("This program will create two processes\n\n");

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

sleep(2);

if ((pid1 = fork()) > 0){

printf("Parent: successfully created my first child, pid=%d\n", (int)pid1);

sleep(1);

if ((pid2 = fork()) > 0){

printf("Parent: successfully created my second child, pid=%d\n",(int)pid2);

sleep(5);

printf("\n\nParent: Bye Bye\n");

exit(0);

}

else

if(pid2==0){

sleep(3);

printf("\n\n");

printf("Child 2:my pid is %d, my ppid is %d\n",(int)getpid(),

(int)getppid());

printf("\n\nChild 2: Bye Bye\n");

}

}

else{

sleep(2);

printf("\n\n");

printf("Child 1:my pid is %d, my ppid is %d\n",(int)getpid(),(int)getppid());

printf("\n\nChild 1: Bye Bye\n");

}

exit(0);

}

**Question 2: Write a C program to simulate the following scenario:**

**1- The parent opens an existing file, for example "file.txt", using the**

**system call open().**

**2- The parent creates a child using the system call fork().The parent calls the system call wait() to wait for its child process to terminate**

**3- The child uses the file descriptor, returned to the parent by open(),**

**to display on the screen the contents of "file.txt". This task should**

**be done using the read()/write() system calls.**

**4- The child sleeps for 5 seconds, prints the message "Child terminating",**

**sleeps for another 5 seconds and, finally calls the exit(20).**

**5- The parent, once awake, prints the message "My child has terminated"**

**and print the status value returned by the child.The parent terminates**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

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

pid\_t pid;

int fd;

int status;

char c;

if(argc !=2 ){

printf("Call model: %s <inputFile>\n", argv[0]);

exit(0);

}

if ( (fd = open(argv[1], O\_RDONLY)) == -1){

printf("%s could not be opened, please create file first\n", argv[1]);

exit(0);

}

printf("Parent is going to create a child process\n\n");

sleep(1);

if ((pid = fork()) > 0){

printf("Parent: child created, waiting for my child to terminate\n\n");

wait(&status);

printf("Parent: child terminated with status=%d\n", WEXITSTATUS(status));

printf("Parent: Bye Bye\n");

}

else

if(pid==0){

sleep(2);

printf("Child: I am going to disply the file contents\n");

sleep(2);

while(read(fd, &c, 1)>0)

write(STDOUT\_FILENO, &c, 1);

sleep(2);

printf("\n\nChild: I am done, terminating with status = 20\n\n");

sleep(2);

exit(20);

}

exit(0); }

**Question 3: Write a C program to simulate the following scenario:**

**1- The parent creates a file, call it "newFile.txt", and opens an existing**

**file, for example "file.txt", using the system call open().**

**2- The parent creates a child using the system call fork()**

**3- The parent calls the system call wait() to wait for its child process to**

**terminate**

**4- The child uses the file descriptors, returned to the parent by open(),**

**to read the contents of "file.txt" and write it to the file "newFile.txt".**

**5- The child sleeps for 5 seconds, writes the message "Child terminating" to**

**"newFile.txt", sleeps for another 5 seconds and, finally calls the exit(20).**

**6- The parent, once awake, writes to "Newfile.txt" the message**

**"My child has terminated" and the status value returned by the child.**

**7- The parent terminates**

**8- Check the contents of "newFile.txt"**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/wait.h>

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

pid\_t pid;

int fd1, fd2;

int status;

char ar[100], c;

if(argc !=3 ){

printf("Call model: %s <inputFile> <outputFile>\n", argv[0]);

exit(0);

}

if ( (fd1 = open(argv[1], O\_RDONLY)) == -1){

printf("%s could not be opened, please create file first\n", argv[1]);

exit(0);

}

if ( (fd2 = open(argv[2], O\_WRONLY|O\_CREAT|O\_TRUNC, 0700)) == -1){

printf("%s could not be created, please check directory\n", argv[2]);

exit(0);

}

printf("Parent is going to create a child process\n\n");

sleep(1);

if ((pid = fork()) > 0){

strcpy(ar, "Parent: child created, waiting for my child to terminate\n\n");

write(fd2, ar, strlen(ar));

wait(&status);

strcpy(ar, "Parent: child terminated with status=");

write(fd2, ar, strlen(ar));

sprintf(ar,"%d\n", WEXITSTATUS(status));

write(fd2, ar, strlen(ar));

strcpy(ar, "Parent: Bye Bye\n");

write(fd2, ar, strlen(ar));

}

else

if(pid==0){

sleep(2);

strcpy(ar, "Child: I am going to disply the file contents\n");

write(fd2, ar, strlen(ar));

sleep(2);

while(read(fd1, &c, 1)>0)

write(fd2, &c, 1);

sleep(2);

strcpy(ar, "\n\nChild: I am done, terminating with status = 20\n\n");

write(fd2, ar, strlen(ar));

sleep(2);

exit(20);

}

exit(0);

}