Operating System Lab-2

Process creation/ Inter process communication (IPC)

Process Creation and Managemen

Patil Amit Gurusidhappa

19104004

B11

Q1.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    fork();

    printf("Hello world!\n");

    return 0;

}

Q2

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

int main()

{

    pid\_t cpid;

    int stat;

    if (fork() == 0) // on success fork return 0 otherwise -1

        exit(10); // exit status code set to 10

    else

        cpid = wait(&stat);

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

    printf("Child pid = %d\n", cpid);

    if (WIFEXITED(stat))

        printf("Exit status: %d\n", WEXITSTATUS(stat));

    return 0;

}

/\*

Parent pid = 6652

Child pid = 16666

Exit status: 10

 \*/

Q3

#include <stdio.h>

#include <unistd.h>

int main()

{

    char \*args[] = {"./q3exac", NULL};

    execvp(args[0], args);

    printf("%s", args[0]);

    printf("I'm  executing cause  command told me to do so");

    return 0;

}

Q4

#include <stdio.h>

#include <unistd.h>

int main()

{

    int pipefds[2]; // count of pipe file descriptiors

    int returnstatus;

    char writemessages[2][20] = {"Hi", "There"};

    char readmessage[20];

    returnstatus = pipe(pipefds);

    if (returnstatus == -1)

    {

        printf("Unable to create pipe");

        return 0;

    }

    printf("Writing to pipe message %s \n", writemessages[0]);

    write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

    read(pipefds[0], readmessage, sizeof(readmessage));

    printf("%s", readmessage);

    printf("\n");

    printf("Writing to pipe message %s \n", writemessages[1]);

    write(pipefds[1], writemessages[1], sizeof(writemessages[1]));

    read(pipefds[0], readmessage, sizeof(readmessage));

    printf("%s", readmessage);

    return 0;

}

/\*

Writing to pipe message Hi

Hi

Writing to pipe message There

 \*/

Q5

#include <stdio.h>

#include <unistd.h>

int main()

{

    int pipefds[2];

    int returnstatus;

    int pid;

    char writemessage[2][20] = {"Hi", "Hello"};

    char readmessage[20];

    returnstatus = pipe(pipefds);

    if (returnstatus == -1)

    {

        printf("Unable to create pipe");

        return 1;

    }

    pid = fork();

    // child process

    if (pid == 0)

    {

        read(pipefds[0], readmessage, sizeof(readmessage));

        printf(" Child process from pipe message 1 %s \n", readmessage);

        read(pipefds[0], readmessage, sizeof(readmessage));

        printf(" Child process from pipe message 2 %s \n", readmessage);

    }

    else

    {

        printf("writing msg 1 to pipe %s \n", writemessage[0]);

        write(pipefds[1], writemessage[0], sizeof(writemessage[0]));

        printf("writing msg 2 to pipe %s \n", writemessage[1]);

        write(pipefds[1], writemessage[1], sizeof(writemessage[1]));

    }

    return 0;

}

/\*

 writing msg 1 to pipe Hi

 writing msg 2 to pipe Hello

 Child process from pipe message 1 Hi

 Child process from pipe message 2 Hello

 \*/

Q6

#include <stdio.h>

#include <unistd.h>

int main()

{

    int pipefds1[2];

    int pipefds2[2];

    int pid;

    int returnstatus1;

    int returnstatus2;

    char pipe1writemessage[20] = "Hi";

    char pipe2writemessage[20] = "Hello";

    char readmessage[20];

    returnstatus1 = pipe(pipefds1);

    if (returnstatus1 == -1)

    {

        printf("Not able create pipe 1");

    }

    returnstatus2 = pipe(pipefds2);

    if (returnstatus2 == -1)

    {

        printf("Not able create pipe 2");

    }

    pid = fork();

    if (pid != 0) // Parent process

    {

        close(pipefds1[0]); // Close the unwanted pipe1 read side

        close(pipefds2[1]); // Close the unwanted pipe2 write side

        printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);

        write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));

        read(pipefds2[0], readmessage, sizeof(readmessage));

        printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);

    }

    else

    {                       //child process

        close(pipefds1[1]); // Close the unwanted pipe1 write side

        close(pipefds2[0]); // Close the unwanted pipe2 read side

        read(pipefds1[0], readmessage, sizeof(readmessage));

        printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);

        printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

        write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

    }

    return 0;

}