HW1(Meng Xian, Du - CBB108047)

Advanced Operation System, Spring 2023

Q1

Write a program that calls fork(). Before calling fork(), have the main process access a variable (e.g., x)and set its value to something (e.g., 100).
What value is the variable in the child process?
What happens to the variable when both the child and parent change the value of x?

Solution: Please refer to q1.c

q1.c :

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    int x = 100;
    printf("X = %d \n", x);
    printf("pid = %d \n", getpid());
    int pid = fork();
    if (pid < 0)
    {
        fprintf(stderr, "Oh my goodman, fork is fail\n");
        exit(1);
    }
    else if (pid == 0)
    {
        x = 1;
        printf("Oh my friend, I am child process.\t");
        printf("My PID is %d and Now x = %d\n", getpid(), x);
    }
    else
    {
        x = 2;
        printf("Oh my child, I am parent process.\t");
        printf("My PID is %d and Now x = %d\n", getpid(), x);
        wait(NULL);
        printf("\n--------------\n");
    }
}
```

Q1 Result

```
X = 100
pid = 24854
Oh my child, I am parent process.       My PID is 24854 and Now x = 2
Oh my friend, I am child process.       My PID is 24855 and Now x = 1

--------------
```

Anwser

As we can see in the line 8 of q1.c, we have declared a variable “x” with value 100.
As we can see that the line 11 we fork a child process and save it’s return value to an integer variable pid. If it fail when it fork the program will run line 12~13.
If the pid is 0. that stand for the process is child process, then the process executing line 19 to 21 reassign x to 1, Also print some message and pid.
If the pid more then 0, the program run line 25 to 29. To reassign x to 2 and print some message.Finaly wait for subprocess reutrn then program exit.
In the execution result you can see the variable in child process is assign to 1 and the variable in parent process is assign to 2.
If the both process child and parent change the variable, the variable will change by child first then change by parent if parent process has used wait function.

Q2

Write a program that opens a file (with the open() system call)and then calls fork() to create a new process.
Can both the childand parent access the file descriptor returned by open()?
What happens when they are writing to the file concurrently, i.e., at the same time?

Solution : Please refer to q2.c

q2.c

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <alloca.h>
#define bool int
#define true 1
#define false 0


const char filename[] = "q2.txt";
void err();
int parent();
int child();
int openFile();
int writeFile();
char *getTypeStr(char *type);
char *readfile(char *type);
int main()
{
    remove(filename);
    printf("Program Start. pid = %d\n",getpid());

    int pid = fork();

    if (pid < 0)
    {
        err();
    }
    else if (pid == 0)
    {
        if (!writeFile("Child"))
        {
            return 0;
        }
        if (!strcmp(readfile("Child"), ""))
        {
            return 0;
        }
    }
    else
    {
        if (!writeFile("Parent"))
        {
            return 0;
        }
        if (!strcmp(readfile("Parent"), ""))
        {
            return 0;
        }
    }
}

void err()
{
    fprintf(stderr, "Oh my goodman, fork is fail\n");
    exit(1);
}
/// @brief To open the file.
/// @param type
/// @param readable if true => read,false =>write
/// @return
int openFile(char *type, bool readable)
{
    printf("(%s)Opening the file. The process. My PID = %d \n", type, (int)getpid
    int file_des = -1;
    if (readable)
    {
        file_des = open(filename, O_RDONLY | O_CREAT, S_IRWXU);
    }
    else
    {
        file_des = open(filename, O_WRONLY | O_CREAT|O_APPEND , S_IRWXU);
    }
    if (file_des < 0)
    {
        printf("(%s)File open error with open return %d\n", type, file_des);
```

```
        printf("(%s)File open error with open return %d\n", type, file_des);
        return -1;
    }
    printf("(%s)The file has been opened.(file_des = %d)\n", type, file_des);
    return file_des;
}

int writeFile(char *type)
{
    int file_des = openFile(type, false);
    if (file_des == -1)
    {
        return 0;
    }
    char* str = getTypeStr(type);
    size_t strLength = strlen(str);
    ssize_t writeStatus = write(file_des, str, strLength);
    printf("(%s)The file is writing. file_des is %d. And the writeStatus is %ld\n", type, file_des, writeStatus);
    if (writeStatus < 0)
    {
        printf("(%s)Writing fail.\n", type);
        return 0;
    }
    printf("(%s)Writing finish. And close the file\n", type);
    close(file_des);
    return 1;
}

char *getTypeStr(char *type)
{
    if (!strcmp("Child", type))
    {
        return "I am Child.\n\0";
    }
    else
    {
        return "I am Parent.\n\0";
    }
}

char *readfile(char *type)
{
    int file_des = openFile(type, true);
    if (file_des < 0)
    {
        return "";
    }
    char buf[2048];
    ssize_t readf = read(file_des, buf, 23);
    if (readf < 0)
    {
        printf("(%s)Read file fail\n", type);
        return "";
    }
    printf("(%s)Reading finish.the content of q2.txt is \n-------\n%s\n-------\n", type, buf);
    close(file_des);
    printf("(%s)The file is closing.\n", type);
    return "";
}
```

Q2 Result

```
Program Start. pid = 28407
(Parent)Opening the file. The process. My PID = 28407
(Child)Opening the file. The process. My PID = 28408
(Parent)The file has been opened.(file_des = 3)
(Parent)The file is writing. file_des is 3. And the writeStatus is 13
(Parent)Writing finish. And close the file
(Child)The file has been opened.(file_des = 3)
(Parent)Opening the file. The process. My PID = 28407
(Child)The file is writing. file_des is 3. And the writeStatus is 12
(Child)Writing finish. And close the file
(Child)Opening the fidle. The process. My PID = 28408
(Child)The file has been opened.(file_des = 3)
(Parent)The file has been opened.(file_des = 3)
(Parent)The file is on read. file_des = 3.
(Child)The file is on read. file_des = 3.
(Child)Reading finish.the content of q2.txt is
-------
(Parent)Reading finish.the content of q2.txt is
-------
I am Parent.
I am Parent.
I am Child
-------
I am Child
-------
(Child)The file is closing.
(Parent)The file is closing.
```

Writing at the same time result

```
Program Start. pid = 29022
(Parent)Opening the file. The process. My PID = 29022
(Child)Opening the file. The process. My PID = 29023
(Parent)The file has been opened.(file_des = 3)
(Parent)The file is writing. file_des is 3. And the writeStatus is 13
(Parent)Writing finish. And close the file
(Child)The file has been opened.(file_des = 3)
(Child)The file is writing. file_des is 3. And the writeStatus is 12
(Child)Writing finish. And close the file
(Parent)Opening the file. The process. My PID = 29022
(Child)Opening the file. The process. My PID = 29023
(Child)The file has been opened.(file_des = 3)
(Parent)The file has been opened.(file_des = 3)
(Child)The file is on read. file_des = 3.
(Parent)The file is on read. file_des = 3.
(Parent)Reading finish.the content of q2.txt is
-------
(Child)Reading finish.the content of q2.txt is
-------
I am Parent.
I am Parent.
I am Child
-------
I am Child
-------
(Child)The file is closing.
(Parent)The file is closing.
```

Anwser

- Can both the childand parent access the file descriptor returned by open()?

> Yes, both the process can access the file descriptor that returned by open()
> As you can see in the result line 20 to 24 the read has write and read the file successfully so we can know the string was written and read

- What happens when they are writing to the file concurrently, i.e., at the same time?

> This question I can't answer now because of the experiment I have not finished

Q3

Write another program using fork(). The child process should print "hello"; the parent process should print "goodbye".
You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?

Solution : Please refer to q3.c

q3.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

void proc(int);
void err();
void child();
void parent();
/// @brief To handle the signal event
/// @param sig
void signalHandler(int sig)
{
    sig=sig;
    return ;
}

int main()
{
    /*
        When child proc status has change.
        Signal sent to parent process,
        and call signalHandler to deal with the event.
    */
    signal(SIGCHLD, signalHandler);
    int rc = fork();
    proc(rc);
}
void proc(int rc)
{
    if (rc < 0)
        err();
    if (rc)
    {
        pause();
        parent();
    }
    else
    {
        child();
    }
}
void err()
{
    printf("Something wrong\n");
    return;
}

void parent()
{
    printf("goodbye\n");
}
void child()
{
    printf("hello\n");
}

```

Q3 Result

```
hello
goodbye
```

Answer

- Can you do this without calling wait() in the parent?

> Yes, I do this by using signal.

Q4

Write a program that calls fork() and then calls some form of exec() to run the program /bin/ls. See if you can try all of the variants of exec(), including (on Linux) execl(), execle(), execlp(), execv(), execvp(), and execvpe().
Why do you think there are so many variants of the same basic call?

Solution : Please refer to q4.c

q4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void proc(int, char *[], int);
void err();
void child(char *[], int);
void parent();

const char *path = "/bin/ls";
int main(int argc, char *argv[])
{
    printf("The argc is %d\n", argc);
    int rc = fork();
    proc(rc, argv, argc);
}
void proc(int rc, char *argv[], int argc)
{
    if (rc < 0)
        err();
    if (rc)
    {
        parent();
        wait(NULL);
    }
    else
    {
        child(argv, argc);
    }
}
void err()
{
    printf("Something wrong\n");
    exit(1);
}

void parent()
{
    printf("Parent\n");
}
void child(char *argv[], int argc)
{
    printf("Child\n");
    int i = 0;
    for (i = 0; i < argc; i++)
    {

        printf("%d - ARG : %s\n", i,argv[i]);
        if (strcmp(argv[i], "execl")==0)
        {
            printf("execl\n");
            execl(path, "ls", "-al", NULL);
            return 0;
        }
        if (!strcmp(argv[i], "execlp"))
        {
            printf("execlp\n");
            execlp("ls", "ls", "-al", NULL);
            return 0;
        }
        if (!strcmp(argv[i], "execle"))
        {
            printf("execle\n");
            execle(path, "ls", "-al", NULL,NULL);
            return 0;
        }
        if (!strcmp(argv[i], "execv"))
        {
            printf("execv\n");
            const char *av[] = {"ls", "-l", NULL};
            execv(path, av);
            return 0;
        }
        if (!strcmp(argv[i], "execvp"))
        {
            printf("execvp\n");
            const char *av[] = {"ls", "-l", NULL};
            execvp("ls", av);
            return 0;
```

```
            return 0;
        }
        if (!strcmp(argv[i], "execvpe"))
        {
            printf("execvp\n");
            const char *av[] = {"ls", "-l", NULL};
            execvpe("ls", av,NULL);
            return 0;
        }
    }
```

Q4 Result

```
The argc is 2
Parent
Child
0 - ARG : ./q4
1 - ARG : execvpe
execvp
total 468
-rwxrwxrwx 1 it it 76144 Mar 15 21:05 q1
-rwxrwxrwx 1 it it 82280 Mar 15 22:21 q2
-rwxrwxrwx 1 it it    25 Mar 15 22:21 q2.txt
-rwxrwxrwx 1 it it 75408 Mar 15 22:29 q3
-rwxrwxrwx 1 it it 70624 Mar 15 22:30 q4
-rwxrwxrwx 1 it it 79040 Mar 15 20:58 q7
-rwxrwxrwx 1 it it    37 Mar 15 20:58 q7.txt
-rwxrwxrwx 1 it it 79104 Mar 15 20:58 q8
```

Anwser

- Why do you think there are so many variants of the same basic call?

> Because of C lang does not support function overload. In order to achieve the feature,the designer design the variants.

Q5

Now write a program that uses wait() to wait for the child process to finish in the parent.
What does wait() return? What happens if you use wait() in the child?

Solution : Please refer to q5.c

q5.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

void proc(int);
void err();
void child();
void parent();

int main()
{
    int rc = fork();
    proc(rc);
}
void proc(int rc)
{
    if (rc < 0)
        err();
    if (rc)
    {
        parent();
    }
    else
    {
        child();
    }
}
void err()
{
    printf("Something wrong\n");
    exit(1);
}
void parent()
{
    printf("(Parent) My pid is %d\n", getpid());
    int wait_value = wait(NULL);
    printf("(Parent)The return value of wait is %d\n", wait_value);
    printf("(Parent)So we can know wait function return the pid of child process.\
}
void child()
{
    printf("(Child) My pid is %d\n", getpid());
    int wait_value = wait(NULL);
    printf("(CHild)The return value of wait is %d\n", wait_value);
    printf("\nAccording to the document of wait() said \n\"For errors, return (pid
}
```

Q5 Result

```
(Parent) My pid is 28641
(Child) My pid is 28642
(CHild)The return value of wait is -1

According to the document of wait() said
"For errors, return (pid_t) -1"
 so we can know use wait() in child process will occurs error
 if the child process doesn't have another child process.
(Parent)The return value of wait is 28642
(Parent)So we can know wait function return the pid of child process.
```

Anwser

- What does wait() return? What happens if you use wait() in the child?

> wait function return an integer aka pid that child process which is the first exit

Q6

Write a slight modification of the previous program, this time using waitpid() instead of wait().
When would waitpid() be useful?

Solution : Please refer to q6.c

q6.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>

void proc(int);
void err();
void child(int);
void parent(int);

int main()
{
    int rc = fork();
    proc(rc);
}
void proc(int rc)
{
    if (rc < 0)
        err();
    if (rc)
    {
        parent(rc);
    }
    else
    {
        child(rc);
    }
}
void err()
{
    printf("Something wrong\n");
    exit(1);
}
void parent(int rc)
{
    int status;
    printf("(Parent) My pid is %d\n", getpid());
    int wait_value = waitpid(rc,&status,0);
    printf("(Parent)The return value of wait is %d\n", wait_value);
}
void child(int rc)
{
    int status;
    printf("(Child) My pid is %d\n", getpid());
    int wait_value = waitpid(rc,&status,WNOHANG);
    printf("(CHild)The return value of wait is %d\n", wait_value);
}
```

Q6 Result

```
(Parent) My pid is 28706
(Child) My pid is 28707
(CHild)The return value of wait is -1
(Parent)The return value of wait is 28707
```

Anwser

- When would waitpid() be useful?

> if you have many child process and you want to wait for the specific child process you can use waitpid to wait for the process end the execution

Q7

Write a program that creates a child process, and then in the child closes standard output (STDOUT FILENO). What happens if the child calls printf() to print some output after closing the descriptor?

Solution : Please refer to q7.c

q7.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

void proc(int);
void err();
void parent();
void child();

int main()
{
    int pid = fork();
    proc(pid);
    if (pid == getpid())
    {
        printf("\n---------------Program END---------------\n");
    }
}
void proc(int pid)
{
    switch (pid)
    {
    case 0:
        child();
        break;
    case -1:
        err();
        break;

    default:
        parent();
        break;
    }
}
void err()
{
    printf("Error");
    exit(-1);
}

void parent()
{
    printf("(Parent)I am parent\n");
    wait(NULL);
    printf("(Parent)Parent END\n");
}
void child()
{
    printf("(Child)Before CLOSE\n");
    close(STDOUT_FILENO);
    open("q7.txt", O_RDWR | O_TRUNC | O_CREAT, S_IRWXU);
    printf("(Child)After CLOSED\n");
    printf("(Child)Child END\n");
}

```

Q7 Result

```
(Parent)I am parent
(Child)Before CLOSE
(Parent)Parent END
```

Anwser

- What happens if the child calls printf() to print some output after closing the descriptor?

> The output message that general after closing the descriptor will disapeared because the value in the file description table 1 is null and the message throw to the null pointer is invalid to print out.

Q8

Write a program that creates two children, and connects the standard output of one to the standard input of the other,using the pipe() system call.

Solution : Please refer to q8.c

q8.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

void err()
{
    fprintf(stderr, "Fork err");
    exit(1);
}

int main()
{
    char content[50]="";
    int p[2];
    if (pipe(p) < 0)
    {
        perror("Pipe err");
        exit(1);
    }
    int pid = fork();
    if (pid < 0)
        err();
    if (!pid)
    {
        printf("Hello Sub1\n");
        int pid2 = fork();
        if (pid2 < 0)
            err();
        if (!pid2)
        {
            printf("Hello sub2\n");
            strcpy(content,"(Child2)Hello from sub 2");
            write(p[1],content,50);
            return ;
        }
        else {

        }
        wait(NULL);
        char ct[50];
        read(p[0],ct,50);
        printf("(Child1)%s\n",ct);
    }
    else
    {
        printf("This is parent\n");
        wait(NULL);
    }
    return 1;
}

```

Q8 Result

```
This is parent
Hello Sub1
Hello sub2
(Child1)(Child2)Hello from sub 2
```