OS-assignment1

221041035 ZHANG Xinran

October 2021

1 TASK1

1.1 code

First of all, we set a function to handle the value of signal. When signal == SIGCHLD, print "Parent process receives SIGCHLD signal" and return to null.

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

void handle(int sig) {
    if (sig == SIGCHLD) {
        printf("Parent process receives SIGCHLD signal.\n");
    }
}
```

Also, we do some format transformation and set some variables. In the new variable–'arguments', we fill with the items in argv[ ].

```
int main(int argc, char* argv[]) {
    signal(SIGCHLD, handle);
    pid_t fpid;
    int status;
    char* arguments[100];
    if (argc > 1) {
        for (int i = 1; i < argc; ++i) {
            arguments[i - 1] = argv[i];
        }
    }
    arguments[argc - 1] = NULL;
```

After this processing, we can proceed to a series of operations in the assignment.

```c
/* fork a child process */
    printf("Process start to fork\n");
    fpid = fork();
    if (fpid < 0) {
        printf("Failed.\n");
    }
    else if (fpid == 0) {
        printf("I'm the Child Process, my pid = %d\n", getpid());
        printf("Child process start to execute test program:\n");
        /* execute test program */
        execv(argv[1], arguments);
    }
    else {
        printf("I'm the Parent Process, my pid = %d\n", getpid());

        /* wait for child process terminates */
        wait(&status);

        /* check child process' termination status */
        if (status == 0) {
            printf("Normal termination with EXIT STATUS = 0\n");
        }
        else {
            printf("CHILD PROCESS STOPPED, BUT EXIT STATUS = %d\n", status);
        }
    }
}
```

1.2 detailed process

Firstly, we set pid_t fpid= fork() to follow the fork. By checking the value of fpid, we can find whether the fork process works well.

If fpid<0, which means that there is sth wrong, we print "failed".

After fpid=fork(), there are 2 processes now. One is called child process, and one is called parent process. They will run the following codes relatively and gives some outputs.

If fpid==0, then we know that it is the child process and print the pid of this child process. Also, let the child process start to execute test program.

If fpid!=0, we know it is the parent process. First, we print the pid of parent process. Then, we wait for child process terminates and test its status. If the status==0, it means there is a normal termination. While is the status!=0, it means that the child

process broke down.

1.3 output

In the terminal, we input 'make' and get this output.

```
hadoop@ubuntu:~/Documents/OS/assignment_1_221041035/source1/source/program1$ make
cc -o abort abort.c
cc -o alarm alarm.c
cc -o bus bus.c
cc -o floating floating.c
cc -o hangup hangup.c
cc -o illegal_instr illegal_instr.c
cc -o interrupt interrupt.c
cc -o kill kill.c
cc -o normal normal.c
cc -o pipe pipe.c
cc -o program1 program1.c
cc -o quit quit.c
cc -o segment_fault segment_fault.c
cc -o stop stop.c
cc -o terminate terminate.c
cc -o trap trap.c
```

Figure 1: make

Then, finally, we compile the program1.c and get the output we need.

If the program in child process has a normal termination, the output is like this.

```
hadoop@ubuntu:~/Documents/OS/assignment_1_221041035/source1/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 23253
I'm the Child Process, my pid = 23254
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receives SIGCHLD signal.
Normal termination with EXIT STATUS = 0
```

Figure 2: normal termination

If the child process stopped abnormally, the output is like this.

```
hadoop@ubuntu:~/Documents/OS/assignment_1_221041035/source1/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 23498
I'm the Child Process, my pid = 23499
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal.
CHILD PROCESS STOPPED, BUT EXIT STATUS = 135
```

Figure 3: child process stopped

2 TASK2 & Bonus

Because I am a student in master of data science without any basis in computer science (I just learned a little python), I only finished the first task.

In the learning process, I independently configured the virtual machine and the programming tools (VScode) for the Linux environment.

In my opinion, the bonus question can be solve by a function, which can scan the directory. But I failed to accomplish the whole program.

I will learn harder in the following several months and tried to solve the task 2 and bonus problem.